PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: Philippe Armangau, et al.	
Serial No.: 10/603,411 Confirm: 4172	Group Art Unit: 2168
Filed: 06/25/2003	Examiner: Oni, Olubusola
For: Replication of Snapshot Using a File System Copy Differential	Atty. Dkt. No.: EMCR:0095NPU
	Technology Center 2100

APPEAL BRIEF TO THE BOARD OF PATENT APPEALS AND INTERFERENCES

Commissioner for Patents
PO Box 1450
Alexandria, Virginia 22313-1450

Sir:

This Appeal Brief is in support of the Notice of Appeal filed April 10, 2007 from the decision of the Examiner in the Final Official Action dated Jan. 10, 2007. Please apply the previously paid appeal brief fee to this new appeal brief. Please deduct any deficiency in any required fee from EMC Corporation Deposit Account No. 05-0889.

I. REAL PARTY IN INTEREST

The real party in interest is EMC Corporation, by virtue of an assignment recorded at Reel 014252 Frame 0875.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

III. STATUS OF THE CLAIMS

Claims 1-66 have been presented for examination.

Claims 1-7, 10-15, 19-32, 35-40, 44-53, 55-58, and 62-66 have been cancelled.

Claims 8-9, 16-18, 33-34, 41-43, 54, and 59-61 have been finally rejected, and are being appealed.

IV. STATUS OF AMENDMENTS

No amendment was filed after the final Official Action.

V. SUMMARY OF CLAIMED SUBJECT MATTER

The appellants' invention of claim 8 provides a method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility receives a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies. The snapshot copy facility responds to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies. (Appellants' specification, page 3, lines 4-11.) The snapshot copy facility has an index for each snapshot copy for indicating changes between said each snapshot copy and a next snapshot copy of the production file system. (Appellants' specification, page 3, lines 15-17.) For example, as shown in appellants' FIG. 25 and described in appellants' specification on page 62, lines 7-15, the index for each snapshot copy is a respective bit map of the snapshot copy facility of FIG. 25. The method of appellants' claim 8 includes scanning the indices for a sequence of the snapshot copies including the index for the specified older one of the snapshot copies and a respective index for each of a plurality of snapshot copies of the production file system that are both younger than the specified older one snapshot copies and older than the specified younger one of the snapshot copies. (Appellants' specification, page 3, lines 17-22.) For example, as shown in appellants' FIGS. 34 and 35, reproduced below, and described in the appellants' specification on page 63 lines 16-22, the indices for the sequence of the snapshot copies are scanned by a

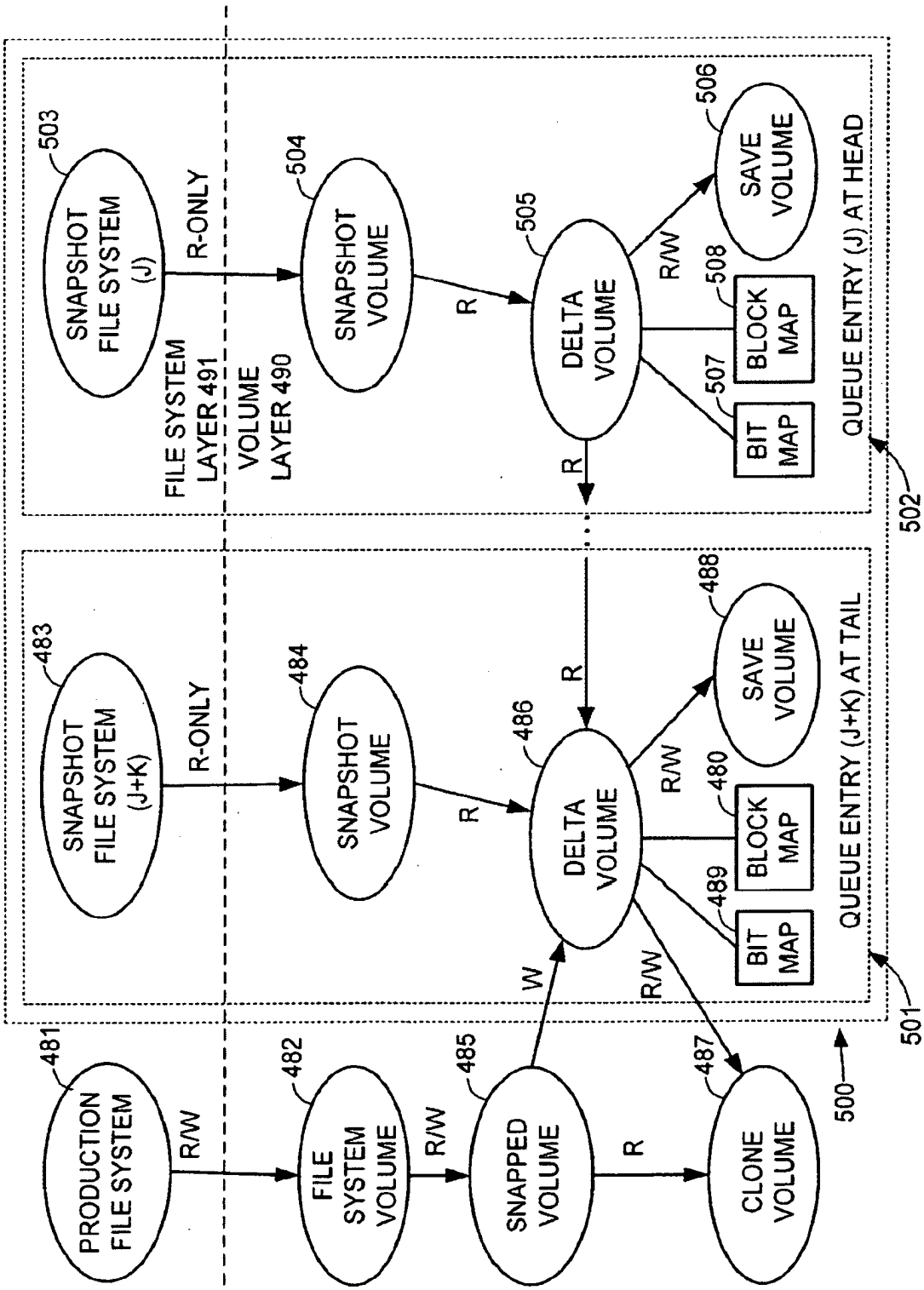


FIG. 25 (PRIOR ART)

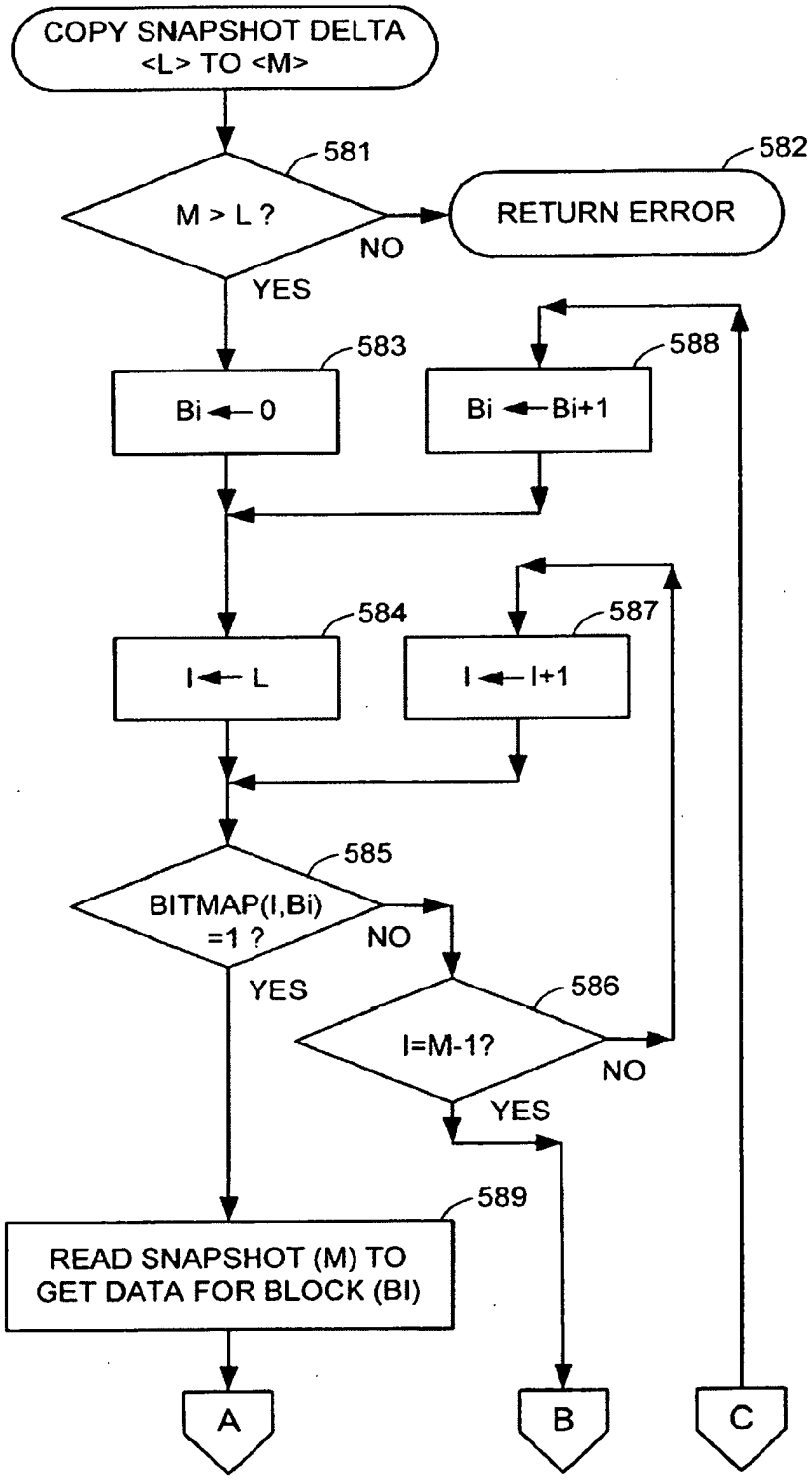

COPY SNAPSHOT DELTA <L> TO <M>
581
M > L ?
NO
582
RETURN ERROR
YES
583
Bi ← 0
588
Bi ← Bi+1
584
I ← L
587
I ← I+1
585
BITMAP(I,Bi) =1 ?
NO
YES
586
I=M-1?
NO
YES
589
READ SNAPSHOT (M) TO GET DATA FOR BLOCK (BI)
A
B
C

FIG. 34

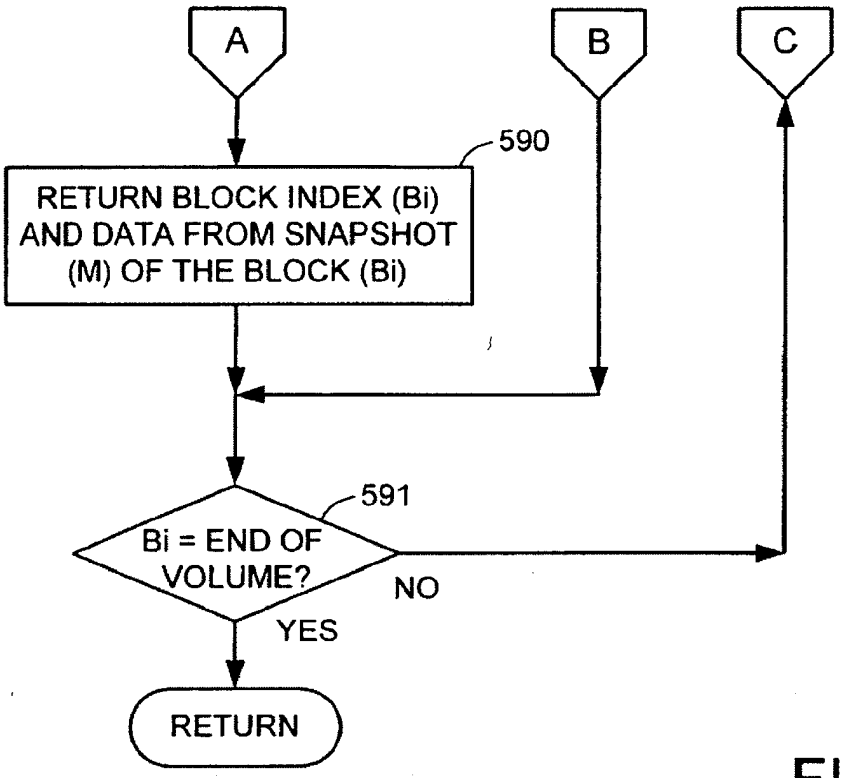


FIG. 35

program routine having an outer loop indexing blocks of data in the file system, and an inner loop indexing the snapshot copies in the sequence of the snapshot copies. "The program in the flowchart of FIGS. 34-35 has an inner loop including steps 585, 586, 587 that indexes the snapshots L to snapshot M-1. This sequence includes the snapshot L and the snapshots that are both younger than the snapshot L and older than the snapshot M. The program in the flowchart of FIGS. 34-35 has an outer loop including steps 584, 585, 586, 591, and 588 that indexes the blocks. When a bit in the indexed bit map is found to be set in step 585, the inner loop is exited to return the block index (B_i) and the data in the snapshot M for block (B_i)."

Appellants' claim 33 defines a snapshot copy facility including storage for storing a plurality of snapshot copies of a production file system, and at least one processor programmed for receiving a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies; and for responding to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies, using the method as specified in appellants' claim 8. See appellants' specification, page 5, line 21 to page 6, line 6; page 6 line 10 to line 18; appellants' FIG. 25 and appellants' specification on page 62, lines 7-15; and appellants' FIGS. 34-35 and appellant' specification, page 63, lines 11-22, as summarized above.

Appellants' claim 54 defines a program storage device containing a program for a snapshot copy facility, the snapshot copy facility storing a plurality of snapshot copies of a production file system, wherein the program is executable for responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies, using the method as specified in appellants' claim 8. See appellants' specification, page 8, lines 3-10 and 14-22; appellants' FIG. 25 and appellants' specification on page 62, lines 7-15; and appellants' FIGS. 34-35 and appellants' specification, page 63, lines 11-22, as summarized above.

Appellants' invention of claim 9 provides a method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy

facility receives a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies. The snapshot copy facility responds to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies. (Appellants' specification, page 3, lines 4 to 11.) The snapshot copy facility has an index for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy. For example, as shown in appellants' FIG. 42, as reproduced below, a snapshot copy facility has a meta bit map 691 for the production file system 481 and a respective meta bit map 692, 693 for each snapshot file system 483, 503) for each snapshot copy. (Appellants' specification, page 71, lines 12-14.) The meta bit map for the production file system has a bit for indicating whether or not each allocated block of storage in the production file system is valid or not. (Appellants' specification, page 70, lines 6-12.) Whenever a snapshot copy of the production file system is created, a snapshot copy of the meta bit map is also created. (Appellants' specification, page 71, lines 17-21.) The method of appellants' claim 9 further includes scanning the index for the specified younger one of the snapshot copies, and when the index indicates that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies. The is shown in appellants' FIG. 40, as reproduced below, and as described in appellants' specification on page 68 line 11 to page

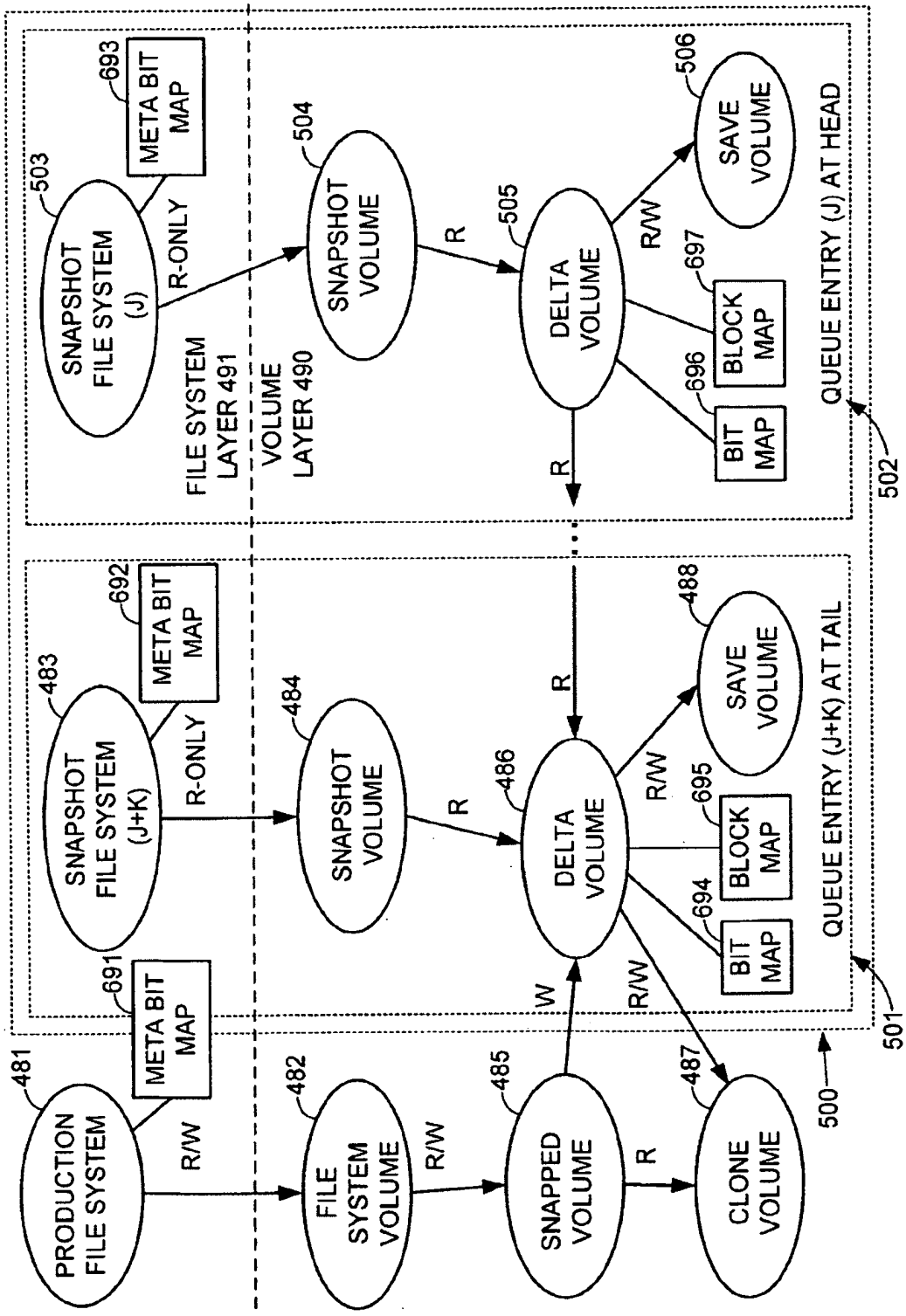

PRODUCTION FILE SYSTEM
481
R/W
META BIT MAP
691
FILE SYSTEM VOLUME
482
R/W
SNAPSHOT FILE SYSTEM (J+K)
483
R-ONLY
META BIT MAP
692
SNAPSHOT VOLUME
484
R
SNAPPED VOLUME
485
W
R/W
DELTA VOLUME
486
R
R/W
SAVE VOLUME
488
BLOCK MAP
695
BIT MAP
694
CLONE VOLUME
487
R
SNAPSHOT FILE SYSTEM (J)
503
R-ONLY
META BIT MAP
693
SNAPSHOT VOLUME
504
R
DELTA VOLUME
505
R
R/W
SAVE VOLUME
506
BLOCK MAP
697
BIT MAP
696
FILE SYSTEM LAYER 491
VOLUME LAYER 490
QUEUE ENTRY (J+K) AT TAIL
501
QUEUE ENTRY (J) AT HEAD
502
500

FIG. 42

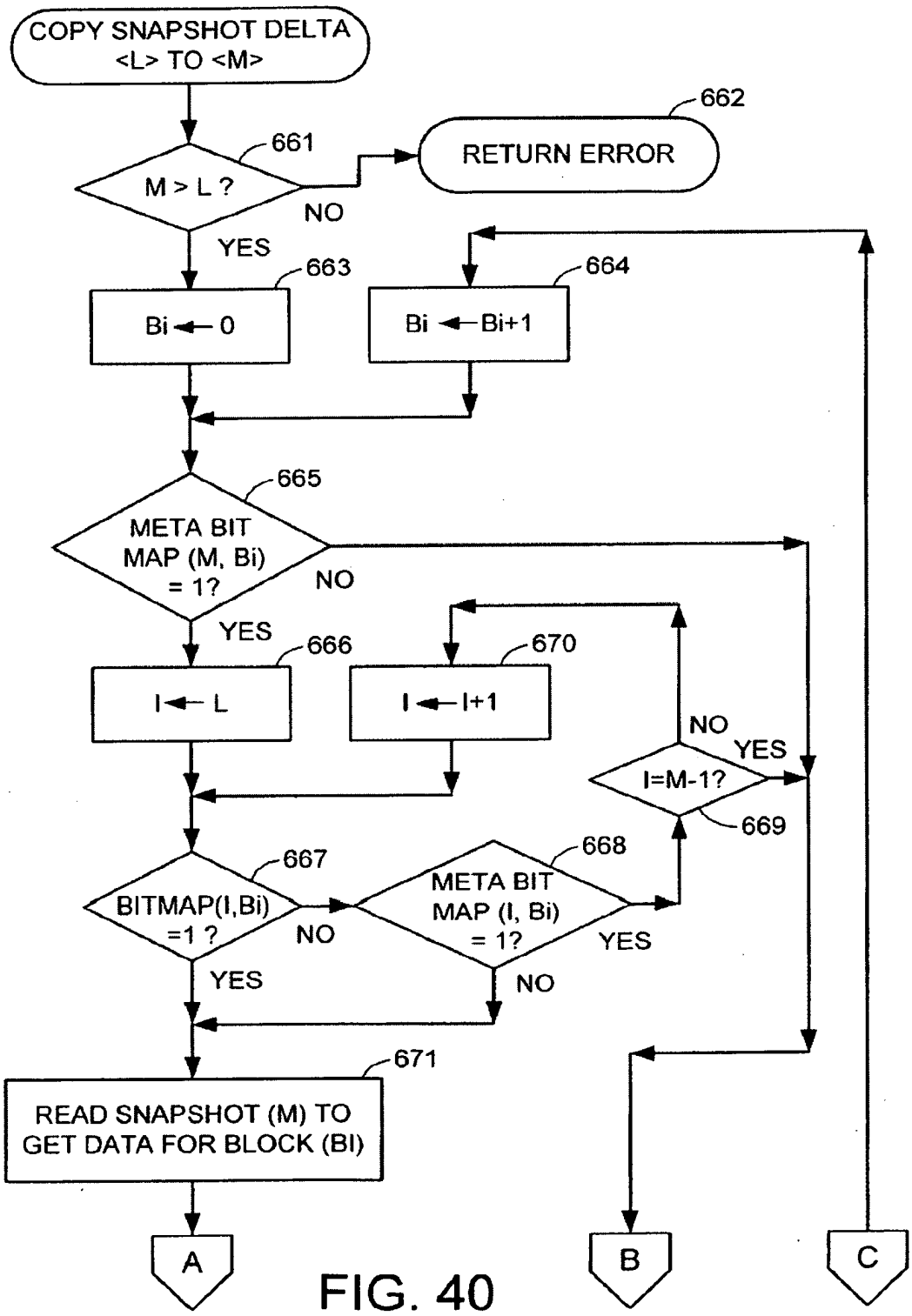

COPY SNAPSHOT DELTA <L> TO <M>
661
M > L ?
NO
662
RETURN ERROR
YES
663
Bi ← 0
664
Bi ← Bi+1
665
META BIT MAP (M, Bi) = 1?
NO
YES
666
I ← L
670
I ← I+1
NO
YES
I=M-1?
669
667
BITMAP(I,Bi) =1 ?
NO
668
META BIT MAP (I, Bi) = 1?
YES
YES
NO
671
READ SNAPSHOT (M) TO GET DATA FOR BLOCK (BI)
A
B
C

FIG. 40

69 line 5. "In step 665, the meta bit map for snapshot (M) [i.e., for the specified young one of the snapshot copies] has a value for block (B_i) indicating whether or not the block (B_i) is in use for the snapshot (M). In particular, a value of 1 indicates that the block (B_i) is in use for the snapshot (M)." (Appellants' specification, page 68, lines 12-15.) For a value of 1, execution continues to step 666 and then to step 667. "In step 667, if the bit map for snapshot (I) has a value of 1 for the block (Bi), then execution continues to step 671 to read the snapshot (M) to get data for the block (B_i) in order to return the data in response to the command to copy the snapshot delta <L> to <M>." (Appellants' specification, page 68, lines 17 to 20.) In other words, step 667 may determine that the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies. The pertinent change for the block is read in step 671 from snapshot (M). The block index (Bi) and data from snapshot (M) of the block (Bi) are returned in step 590 of FIG. 35. (Appellants' specification, page 63, lines 2-4.)

Appellants' invention of claim 34 provides a snapshot copy facility. The snapshot copy facility includes storage for storing a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility also includes at least one processor programmed for receiving a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies; and for responding to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies. (Appellants' specification, page 5, line 21 to page 6, line 6.) The snapshot

copy facility has an index for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy. As summarized above with respect to claim 16, such a snapshot copy facility is shown in appellants' FIG. 42, and the index for each snapshot copy is a respective meta bit map (692, 693), as described in appellants' specification, page 71, lines 12-14; page 70, lines 6-12; and page 71, lines 17-21. The at least one processor is programmed for scanning the index for the specified younger one of the snapshot copies, and when the index indicates that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies. Thus, the processor is programmed to perform the method of appellants' claim 9, as summarized above and as shown in appellants' FIG. 40 and described in appellants' specification page 68 line 11 to page 69 line 5.

Appellants' invention of claim 16 provides a method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility has an index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system. The method includes scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies. The sequence of the snapshot copies includes the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies. (Appellants'

specification, page 3, lines 12 to 22.) The indices for the sequence of the snapshot copies are scanned by a program routine having an outer loop indexing respective blocks, and an inner loop indexing snapshot copies in the sequence of the snapshot copies. This is shown in appellants' FIG. 40, which is described on page 68 lines 11 to 16 of appellants' specification as including steps similar to steps in appellants' FIG. 39. The outer loop indexing respective blocks includes step 664, and the inner loop indexing the snapshot copies in the sequence of the snapshot copies includes step 670. The snapshot copy facility has a meta bit map for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy. Such a snapshot copy facility is shown in appellants' FIG. 42 and described in appellants' specification, page 71, lines 12-14; page 70, lines 6-12; and page 71, lines 17-21, as summarized above for appellants' claim 9. The method further includes scanning the meta bit map for the specified younger one of the snapshot copies, and when the meta bit map is found to indicate that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies by scanning the indices for the sequence of the snapshot copies. This is shown in appellants' FIG. 40. As summarized above with respect to claim 9, the meta bit map is scanned in step 665 of FIG. 40, and when the meta bit map for the specified younger one (M) of the snapshot copies is a "1" indicating that a block is not known to be invalid, execution continues from step 665 to step 667 to then determine whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies by scanning the indices

(i.e., bitmaps in FIG. 42) in step 667 of FIG. 40 for the sequence of the snapshot copies, as described in appellants' specification on page 68 line 11 to page 69 line 5.

Appellants' invention of claim 41 provides a snapshot copy facility. The snapshot copy facility includes storage for storing a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility includes an index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system. The snapshot copy facility also includes at least one processor programmed for scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies. (Appellants' specification, page 6, lines 7 to 18.) The snapshot copy facility includes a meta bit map for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy. Such a snapshot copy facility is shown in appellants' FIG. 42 and described in appellants' specification, page 71, lines 12-14; page 70, lines 6-12; and page 71, lines 17-21, as summarized above for appellants' claim 9. The at least one processor is programmed for scanning the meta bit map for the specified younger one of the snapshot copies, and when the meta bit map is found to indicate that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of

the snapshot copies by scanning the indices for the sequence of the snapshot copies, as summarized above with respect to appellants' claim 16, and as shown in appellants' FIG. 40 and described in appellants' specification on page 68 line 11 to page 69 line 5.

Appellants' invention of claim 59 provides a program storage device containing a program for a snapshot copy facility. The snapshot copy facility has a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility also has an index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system. The program is executable for scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies. (Appellants' specification, page 8, lines 11-22.) The snapshot copy facility has a meta bit map for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy. Such a snapshot copy facility is shown in appellants' FIG. 42 and described in appellants' specification, page 71, lines 12-14; page 70, lines 6-12; and page 71, lines 17-21, as summarized above for appellants' claim 9. The program storage device is executable for scanning the meta bit map for the specified younger one of the snapshot copies, and when the meta bit map is found to indicate that a block is not known to be invalid, then determining whether the block has changed between the

specified older one of the snapshot copies and the specified younger one of the snapshot copies by scanning the indices for the sequence of the snapshot copies, as summarized above with respect to appellants' claim 16, and as shown in appellants' FIG. 40 and described in appellants' specification on page 68 line 11 to page 69 line 5.

Appellants' invention of claim 17 provides a method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility has a first index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system and that have a "before image" saved for said each snapshot copy. The snapshot copy facility has a second index for each snapshot copy for indicating blocks of data that are not in use in the snapshot copy. The method includes responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by accessing the second index for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices for a sequence of the snapshot copies to determine blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies. The sequence of the snapshot copies includes the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of

the snapshot copies. (Appellants' specification, page 4, lines 1-19.) Such a snapshot copy facility is shown in appellants' FIG. 42. The first index for each snapshot copy (483, 503) is a respective bit map (694, 696), and the second index for each snapshot copy is a respective meta bit map (692, 693). The bit map is described in appellants' specification on page 62, lines 7-15. The meta bit map is described in appellants' specification on page 66, lines 10-12: "In a preferred snapshot copy facility, as described below with reference to FIGS. 41 to 46, there is kept a meta bit map for each snapshot copy for indicating blocks of the production file system that are not used in the snapshot copy." FIGS. 39 and 40 show an example of responding to a request for the difference between a specified older one (L) of the snapshot copies and a specified younger one (M) of the snapshot copies by accessing the second index (meta bit map in step 665 of FIG 40) for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies (see corresponding step 655 in FIG. 39), and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices (in step 667 of FIG. 40) for a sequence of the snapshot copies (e.g., in the inner loop of steps 667, 668, 669, and 670 in FIG. 40) to determine blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, as described in appellants' specification on page 67 line 11 to page 69 line 5.

Appellants' invention of claim 42 provides a snapshot copy facility. The snapshot copy facility includes storage for storing a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in

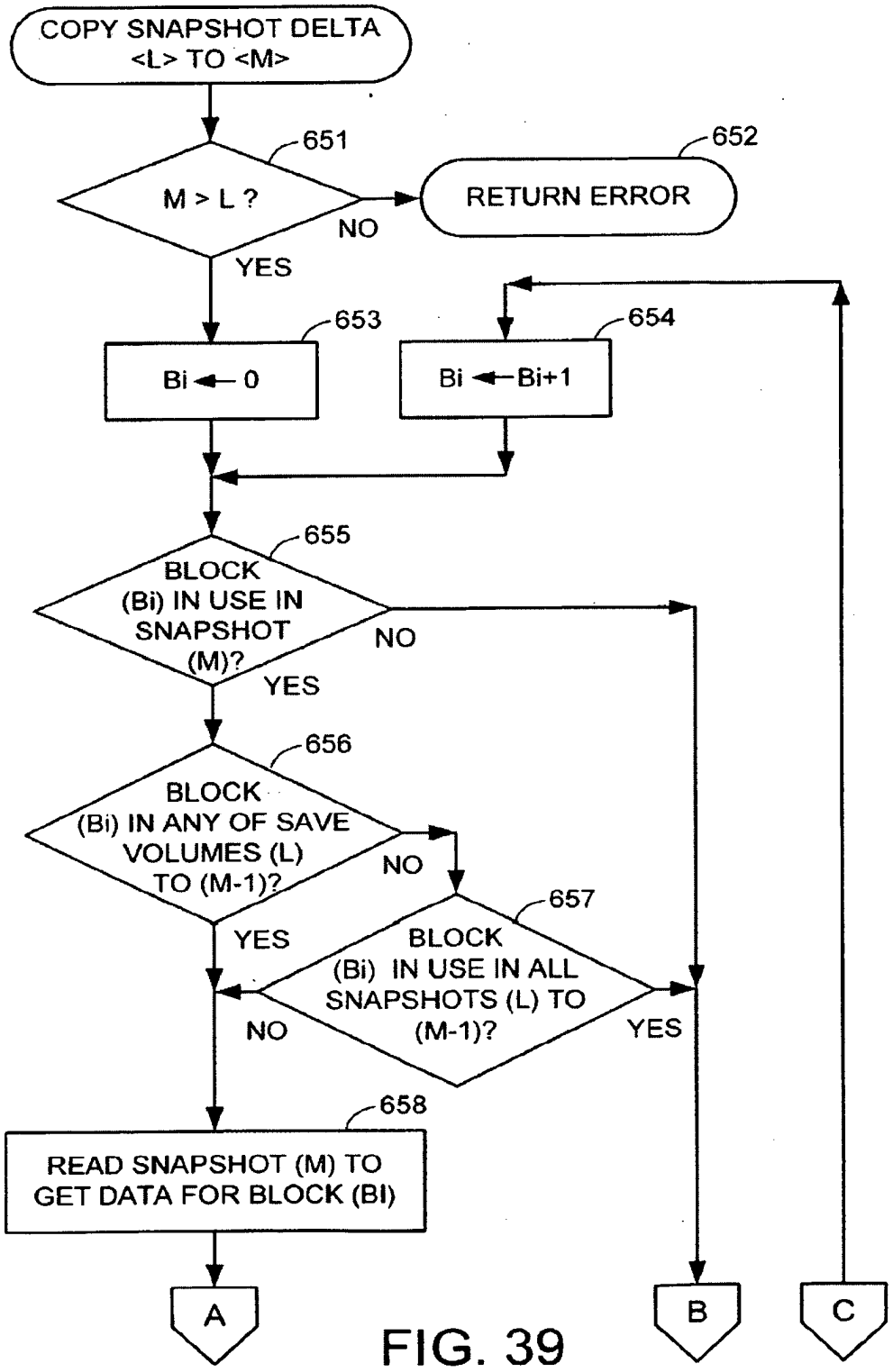

COPY SNAPSHOT DELTA <L> TO <M>
651
M > L ?
NO
652
RETURN ERROR
YES
653
Bi ← 0
654
Bi ← Bi+1
655
BLOCK (Bi) IN USE IN SNAPSHOT (M)?
NO
YES
656
BLOCK (Bi) IN ANY OF SAVE VOLUMES (L) TO (M-1)?
NO
YES
657
BLOCK (Bi) IN USE IN ALL SNAPSHOTS (L) TO (M-1)?
NO
YES
658
READ SNAPSHOT (M) TO GET DATA FOR BLOCK (BI)
A
B
C

FIG. 39

The snapshot copy facility has a first index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system and that have a "before image" for the snapshot copy stored in the storage. The snapshot copy facility has a second index for each snapshot copy for indicating blocks of data that are not in use in the snapshot copy. The snapshot copy facility also has at least one processor programmed for responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by accessing the second index for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices for a sequence of the snapshot copies to determine blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies. The sequence of the snapshot copies includes the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies. (Appellants' specification, page 6, line 19 to page 7, line 15.) Such a snapshot copy facility is shown in appellants' FIG. 42. The first index for each snapshot copy (483, 503) is a respective bit map (694, 696), and the second index for each snapshot copy is a respective meta bit map (692, 693). FIGS. 39 and 40 show an example of how the first index and the second index are used, as described in the appellants' specification and as summarized above with respect to appellants' claim 17.

The invention of appellants' claim 60 provides a program storage device containing a program for a snapshot copy facility. The snapshot copy facility has a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility has a first index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system and that have a "before image" for the snapshot copy stored in the snapshot copy facility. The snapshot copy facility has a second index for each snapshot copy for indicating blocks of data that are not in use in the snapshot copy. The program is executable for responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by accessing the second index for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices for a sequence of the snapshot copies to determine blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies. The sequence of the snapshot copies includes the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies. (Appellants' specification, page 9, lines 1-20.) Such a snapshot copy facility is shown in appellants' FIG. 42. The first index for each snapshot copy (483, 503) is a respective bit map (694, 696), and the second index for each snapshot copy is a respective meta bit map (692, 693).

Appellants' FIGS. 39 and 40 show an example of how a program uses the first index and the second index, as described in the appellants' specification and as summarized above with respect to appellants' claim 17.

Claims 18, 43, and 61 are dependent upon claims 17, 42, and 60, respectively. Each of claims 18, 43, and 61 recites "accessing at least one of the second indices for the snapshot copies in the sequence of the snapshot copies and finding that at least one of the blocks is not in use in at least one of the snapshot copies in the sequence of the snapshot copies to determine that said at least one of the blocks has changed between the older one of the snapshot copies and the younger one of the snapshot copies not changed." Determining that said at least one of the blocks has changed between the older one of the snapshot copies and the younger one of the snapshot copies occurs in steps 656 and 657 of FIG. 39, and more specifically in the inner loop of steps 667, 668, 669, and 670 in FIG. 40. The "accessing at least one of the second indices for the snapshot copies in the sequence of the snapshot copies and finding that at least one of the blocks is not in use in at least one of the snapshot copies in the sequence of the snapshot copies to determine that said at least one of the blocks has changed" occurs in step 668 of FIG. 40 when the meta bit map (I, Bi) is not equal to one, causing execution to branch from step 668 to step 671 in FIG. 40. As described in appellants' specification, page 68 line 22 to page 69 line 4: "In step 668, if the meta bit map for the snapshot (I) does not have a value of 1 for the block (B_i), execution continues to step 671 to read the snapshot (M) to get data for the block (B_i), in order to return the data in response to the command to copy the snapshot delta <L> to <M>. In this case, the block (B_i) is not in use in the snapshot (I)."

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

1. Whether claims 8, 33, and 54 are unpatentable under 35 U.S.C. 102(e) as being anticipated by Goldstein et al. (Pub. No. US 2004/0163009).

2. Whether claims 9, 16-18, 34, 41-43, 59, 60, and 61 are unpatentable under 35 U.S.C. 103(a) over Goldstein et al. (Pub. No. US 2004/0163009) in view of Ohran et al. (Pub. No. US 2002/0112134).

VII. ARGUMENT

1. Claims 8, 33, and 54 are not unpatentable under 35 U.S.C. 102(e) over Goldstein et al. (Pub. No. US 2004/0163009).

"For a prior art reference to anticipate in terms of 35 U.S.C. § 102, every element of the claimed invention must be identically shown in a single reference." Diversitech Corp. v. Century Steps, Inc., 7 U.S.P.Q.2d 1315, 1317 (Fed. Cir. 1988), quoted in In re Bond, 910 F.2d 831,15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990) (vacating and remanding Board holding of anticipation; the elements must be arranged in the reference as in the claim under review, although this is not an ipsis verbis test).

Goldstein discloses a backup apparatus and method suitable for protecting the data volume in a computer system by acquiring a base state snapshot and a sequential series of data volume snapshots. The apparatus concurrently generates succedent and precedent lists of snapshot differences which are used to create succedent and precedent backups respectively. The data volume is restored by overwriting the base state data with data blocks identified in one or more succedent backups. File recovery is accomplished by overwriting data from a concurrent snapshot with one or more precedent backups. (Goldstein, Abstract.)

Goldstein teaches that blocks that have change between snapshots are identified in a list, and are copied into backups and stored in offline storage. (Page 2, paragraph [0024].) A first

succedent backup 131 (B01) is created by copying from the first state snapshot 113 (S_1) all of the data blocks identified in the first succedent snapshot difference list 121. A copy of the snapshot difference list 121 is also included in the first succedent backup 131. (Page 2, paragraph [0030].) Successive precedent backups may be combined into a single precedent backup to reduce offline storage volume and to speed incremental file recovery, as shown in FIG. 10. (Page 4, paragraph [0050].)

In appellants' amendment filed March 23, 2006, claims 8, 33, and 54 were amended to clearly distinguish Goldstein by defining that the sequence of the snapshot copies that is scanned includes the index for the specified older one of the snapshot copies and a respective index for each of a plurality of snapshot copies of the production file system that are both younger than the specified older one snapshot copies and older than the specified younger one of the snapshot copies. Thus, the indices for this particular sequence of the snapshot copies are scanned by a program routine having an outer loop indexing blocks of data in the file system, and an inner loop indexing the snapshot copies in the sequence of the snapshot copies. This is shown in appellants' FIGS. 34 and 35, for example, and described in the appellants' specification on page 63 lines 16-22.

Disclosure in Goldstein pertinent to appellants' claims 8, 33, and 54 is the generation of concatenated precedent lists for example as recited in Goldstein's claim 10 on page 6. The appellants' amendment to claims 8, 33, and 54 clearly define a scanning procedure including inner and outer loops that determine the blocks that have changed over a series of at least three successive snapshots. Instead of generating concatenated precedent snapshot difference lists

between neighboring snapshots in a series by a process of repeated concatenation (e.g., indexing blocks in an inner loop and indexing snapshot copies in an outer loop), the appellants' claims define indexing the snapshot copies in an inner loop and indexing blocks in the outer loop. The appellants' claims call for a scanning procedure directly opposite to what Goldstein would suggest.

In reply to the appellants' amendment and argument, the final Official Action on page 15 cites Goldstein paragraphs [0011], [0024-0033], [0041-0043], and fig. 3, 4, 6 & 7, and concludes "Wherein Goldstein's teachings involve acquiring snapshots of consistent states of data volume, wherein the snapshots are compared to produce a list of blocks that have changed between the snapshots, i.e., the determination of blocks that have changed, thus teaches are synonymous)." However, anticipation under 35 U.S.C. 102 is not satisfied by a teaching of a different way of how to determine blocks that have changed. Instead, for anticipation under 35 U.S.C. 102, every element of the claimed invention must be identically shown in a single reference. See Diversitech Corp. and In re Bond cited above.

2. Claims 9, 16-18, 34, 41-43, 59, 60, and 61 are not unpatentable under 35 U.S.C. 103(a) over Goldstein et al. (Pub. No. US 2004/0163009) in view of Ohran et al. (Pub. No. US 2002/0112134).

The policy of the Patent and Trademark Office has been to follow in each and every case the standard of patentability enunciated by the Supreme Court in Graham v. John Deere Co., 148 U.S.P.Q. 459 (1966). M.P.E.P. § 2141. As stated by the Supreme Court:

> Under § 103, the scope and content of the prior art are to be determined; differences between the prior art and the claims at issue are to be ascertained; and the level of ordinary skill in the pertinent art resolved. Against this background, the obviousness or nonobviousness of the subject matter is determined. Such secondary considerations as commercial success, long felt but unsolved needs, failure of others, etc., might be utilized to give light to the circumstances surrounding the origin of the subject matter sought to be patented. As indicia of obviousness or nonobviousness, these inquiries may have relevancy.

148 U.S.P.Q. at 467.

The problem that the inventor is trying to solve must be considered in determining whether or not the invention would have been obvious. The invention as a whole embraces the structure, properties and problems it solves. In re Wright, 848 F.2d 1216, 1219, 6 U.S.P.Q.2d 1959, 1961 (Fed. Cir. 1988).

It is improper for the Examiner to rely on an applicant's novel discoveries or teachings in his or her own patent application to establish that the invention claimed in that patent application would have been obvious. Riverwood International Corp. v. R.A. Jones & Co., 324 F.3d 1346, 1355, 66 U.S.P.Q.2d 1331, 1338 (Fed. Cir. 2003)("Ones own work may not be considered prior art in the absence of a statutory basis ..."); In re Fritch, 972 F.2d 1260, 1266, 23 U.S.P.Q.2d 1780, 1784 (Fed. Cir. 1992)("It is impermissible to use the claimed invention as an instruction manual or 'template' to piece together the teachings of the prior art so that the claimed invention is rendered obvious."); Orthopedic Equipment Co., Inc. v. United States, 702 F.2d 1005, 1012, 217 U.S.P.Q. 193, 199 (Fed.

Cir. 1983) (It is improper to attempt to establish obviousness by using the applicant's specification as a guide to combining different prior art references to achieve the results of the claimed invention.).

Claims 9 and 34

In paragraph 4 on page 9 of the final Official Action dated Jan. 10, 2007, claims 9, 16-18, 34, 41-43, 59, 60, and 61 were rejected under 35 U.S.C. 103(a) as being unpatentable over Goldstein view of Ohran. Appellants respectfully disagree and submit that that there is insufficient motivation in the prior art as a whole to combine Goldstein and Ohran in the fashion suggested in the Official Action, and the claimed invention would not result from a proper combination of Goldstein and Ohran.

Goldstein is summarized above with respect to applicants' claims 8, 33, and 54.

Ohran discloses a method of restoring a mass storage device, including the corresponding data blocks stored thereon, to a state in which it existed at a prior instant in time to minimize the data loss caused by data blocks becoming corrupt or lost. After a mirrored or backup copy has been made, data blocks that are to be overwritten in response to a write request are stored in a preservation memory prior to being overwritten. The data blocks stored in the preservation memory are time-stamped to designate the chronological order by which the data blocks were overwritten. If data becomes corrupted, the data blocks of the preservation memory are applied to the corrupted data in reverse chronological order until such time that a valid, non-corrupted set

of data is obtained. In this manner, data more recent than that associated with the full mirrored or backup copy can be reconstructed. (Ohran, Abstract.)

Page 11 of the final Official Action says: "Goldstein does not teach 'wherein the snapshot copy facility has a meta bit map for each snapshot copy for indicating blocks of data that are know[n] to be invalid in ..."

Page 11 of the final Official Action cites Oran for "writing invalid or corrupted data to certain data blocks in the mass storage device (See paragraph [0015] and [Fig 3]) ..." Ohran's paragraph 15 (emphasis added) says:

> [0015] In the event that certain data blocks in the mass storage unit device are lost or become corrupted, the data blocks stored in the preservation memory can be used to incrementally restore or reconstruct a valid set of data without reverting completely back to the data as it exists at time T_0. If, for example, invalid or corrupted data is written to certain data blocks in the mass storage device after time T_0, the original, valid data blocks are stored in a preservation memory as described above. Using the time stamps specifying the chronological sequence in which the data blocks stored in the preservation memory were overwritten in the mass storage device, the data blocks in the preservation memory are written to the current data stored in the mass storage device.

Page 11 of the final Official Action concludes that "it would have been obvious at the time of the invention for one of ordinary skill in the art to have modify Goldstein by teachings of Ohran, wherein Ohran's teachings of separating invalid data, while a set of data is eventually

used to reconstruct the invalid data, and been combined with Goldstein's method will enhance checking for change of valid data." Appellants respectfully disagree.

Appellants' claims 9 and 34 define that the snapshot copy facility has an index for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy. Such an index for each snapshot copy is not suggested by a generalized teaching that invalid or corrupted data in mass storage (e.g., a production file system) can be corrected by over-writing them with a backup copy of good data (e.g., a snapshot copy).

In addition, the final Official Action refers to Goldstein as "comparing the based state snapshot and a subsequent series of data volume snapshots to confirm changes, which can only be done if block is valid." (Page 11 of the Final Official Action, emphasis added.) However, there is nothing in Goldstein suggesting that determining a list of snapshot differences requires a determination that the blocks are valid or not in a younger snapshot. Nor is there an inherent need for a meta bit map in a snapshot copy facility, as shown by a comparison of appellants' FIG. 34 for the case of the snapshot copy facility in appellants' FIG. 25, which does not have a meta bit map, to appellants' FIGS. 39 and 40 for the case of the snapshot copy facility in appellants' FIG. 42, which has a meta bit map for each snapshot file system. A retrospective view of inherency is not a substitute for some teaching or suggestion which supports the selection and use of the various elements in the particular claimed combination. In re Rijckaert, 9 F.3d 1531, 1534, 28 U.S.P.Q.2d 1955-1957 (Fed. Cir. 1993)(optimal condition of matching signal time exactly to recording time is not "inherent" in the prior art); In re Newell, 891 F.2d 899, 901, 13 U.S.P.Q.2d

1248, 1250 (Fed. Cir. 1989)(no teaching or suggestion in the prior art that the belt drive of Weiss should be applied to the capstan of an ANSI type of tape cartridge in the manner done by Newell).

In contrast to Goldstein and Ohran, the appellants teach that it is desirable to use a meta bit map so that if the original content of a block is invalid, it need not be saved for a snapshot when new content is written to the block. Not only is less storage used for the snapshot save volume, but also there can be a decrease in the access time for write access to the production file system. A write operation to an invalid block can be executed immediately, without the delay of saving the original contents of the data block to the most recent save volume at the tail of the snapshot queue. (Appellants' specification, page 69, line 8 to page 70, line 5.) In this case, however, a problem arises in that "the bit map for each snapshot (L) indicates whether or not a block has been stored in the save volume for the snapshot (L), and no longer will indicate all of the blocks that have been changed after snapshot (L) and before snapshot (L+1). ... Therefore, for the preferred snapshot copy facility, it is desirable to modify the procedure of FIG. 34 in order to use the information in the meta bit map for the snapshot <M>. In this case, the procedure of FIG. 34 should also be modified to account for the fact that the save volumes no longer store the 'before images' for all of the blocks that may have changed between the successive snapshot volumes." (Appellants' specification, page 66, line 16 to page 67 line 10.)

Furthermore, it is not seen where Ohran teaches checking for blocks not known to be invalid before checking for changes. Instead, Ohran teaches restoring a set of invalid data blocks by incrementally applying data blocks in a preservation memory in reverse chronological order until such time that a valid set of data is obtained. In other words, given that some blocks in a

set have become invalid, Ohran restores the set of blocks by writing "before images" of the blocks in reverse chronological order. This is opposite from the appellants' determining whether a block has changed upon finding that the block is not known to be invalid. Appellants' claim 9, for example, does not recite restoring invalid data blocks. Instead, claim 9 specifically defines a snapshot copy facility responding to a request for the difference between a specified older snapshot copy and a specified younger snapshot copy. The snapshot copy facility responds to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies. The snapshot copy facility has an index for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy. The method includes scanning the index for the specified younger one of the snapshot copies, and when the index indicates that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

In short, responding to a request for the difference between a specified older snapshot copy and a specified younger snapshot copy by determining whether a block has changed between the specified older snapshot copy and a specified younger snapshot copy when an index indicates that the block is not known to be invalid, should not be confused with restoring potentially corrupted data blocks by incrementally applying "before-images" in reverse chronological order until the data is rolled-back to a valid state.

Hindsight reconstruction, using the applicant's specification itself as a guide, is improper because it fails to consider the subject matter of the invention "as a whole" and fails to consider the

invention as of the date at which the invention was made. "[T]here must be some motivation, suggestion, or teaching of the desirability of making the specific combination that was made by the applicant." In re Lee, 277 F.3d 1338, 1343, 61 U.S.P.Q.2d 1430, 1435 (Fed. Cir. 2002) (quoting In re Dance, 160 F.3d 1339, 1343, 48 U.S.P.Q.2d 1635, 1637 (Fed. Cir. 1998)). "[T]eachings of references can be combined only if there is some suggestion or incentive to do so." In re Fine, 837 F.2d 1071, 1075, 5 U.S.P.Q.2d 1596, 1600 (Fed. Cir. 1988) (Emphasis in original) (quoting ACS Hosp. Sys., Inc. v. Montefiore Hosp., 732 F.2d 1572, 1577, 221 U.S.P.Q. 929, 933 (Fed. Cir. 1984)). "[P]articular findings must be made as to the reason the skilled artisan, with no knowledge of the claimed invention, would have selected these components for combination in the manner claimed." In re Kotzab, 217 F.3d 1365, 1371, 55 U.S.P.Q.2d 1313, 1317 (Fed. Cir. 2000). See, for example, Fromson v. Advance Offset Plate, Inc., 755 F.2d 1549, 1556, 225 U.S.P.Q. 26, 31 (Fed. Cir. 1985) (nothing of record plainly indicated that it would have been obvious to combine previously separate lithography steps into one process); In re Gordon et al., 733 F.2d 900, 902, 221 U.S.P.Q. 1125, 1127 (Fed. Cir. 1984) (mere fact that prior art could be modified by turning apparatus upside down does not make modification obvious unless prior art suggests desirability of modification); Ex Parte Kaiser, 194 U.S.P.Q. 47, 48 (PTO Bd. of Appeals 1975) (Examiner's failure to indicate anywhere in the record his reason for finding alteration of reference to be obvious militates against rejection).

Claims 16, 41, and 59.

In the final Official Action, claims 16, 41, and 59 were rejected on the grounds corresponding to the argument in the final Official Action given for rejecting claim 9. Claims 16, 41, and 59 are patentable over Goldstein and Ohran for the same reasons given above for claim 9. In addition, claims 16, 41, and 59 further define that the sequence of the snapshot copies includes the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies, and the indices for the sequence of the snapshot copies are scanned by a program routine having an outer loop indexing respective blocks, and an inner loop indexing snapshot copies in the sequence of the snapshot copies. Thus, claims 16, 41, and 59 further distinguish Goldstein for the reasons given above with respect to claims 8, 33, and 54.

Claims 17, 42, and 60.

With respect to appellants' claims 17, 18, 42, 43, 60, and 61, page 13 of the Final Official Action of Jan. 10, 2007, makes a similar argument with respect to the proposed combination of Goldstein and Ohran for the rejection of claim 9. Therefore, claims 17, 42, and 60 (and their dependent claims 18, 43, and 61) are patentable for the reasons given above for claim 9. In addition, claims 17, 42, and 60 specify a first index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system and that have a "before image" saved for said each snapshot copy, and a second index for said each snapshot copy for indicating blocks of data

that are not in use in said each snapshot copy. Page 14 of the final Official Action says: "Ohran teachings include writing invalid or corrupted data to a certain data block. Wherein the abstract states data loss could be caused by data blocks becoming corrupt or lost, therefore data not in use is equivalent to data loss." The appellants respectfully disagree. In the context of Ohran's reconstruction of corrupted data, it is not understood how one can conclude from the abstract of Ohran that data not in use is equivalent to data loss. There is no need to reconstruct data not in use, but there is a need to reconstruct corrupted data.

Claims 18, 43, and 61

Claims 18, 43, and 61 are dependent on claims 17, 42, and 60, and are therefore patentable over Goldstein and Ohran for the reasons given above with respect to claims 17, 42, and 60. In addition, claims 18, 43, and 61 further define "accessing at least one of the second indices for the snapshot copies in the sequence of the snapshot copies and finding that at least one of the blocks is not in use in at least one of the snapshot copies in the sequence of the snapshot copies to determine that said at least one of the blocks has changed between the older one of the snapshot copies and the younger one of the snapshot copies not changed." For example, determining that said at least one of the blocks has changed between the older one of the snapshot copies and the younger one of the snapshot copies occurs in steps 656 and 657 of appellants' FIG. 39, and more specifically in the inner loop of steps 667, 668, 669, and 670 in FIG. 40. The "accessing at least one of the second indices for the snapshot copies in the sequence of the snapshot copies and finding that at least one of the blocks is not in use in at least one of the snapshot copies in the

sequence of the snapshot copies to determine that said at least one of the blocks has changed" occurs in step 668 of FIG. 40 when the meta bit map (I, Bi) is not equal to one, causing execution to branch from step 668 to step 671 in FIG. 40. This accessing of the at least one of the second indices for the snapshot copies is done "to account for the fact that the save volumes no longer save the "before images" for all of the blocks that may have changed between the successive snapshot copies." (Appellants' specification, page 67, lines 7 to 10.) It is not understood how the further limitations of claims 18, 43, and 61 would have been suggested by Goldstein or Ohran.

In view of the above, the rejection of the appellants' claims should be reversed.

Respectfully submitted,

/ Richard C. Auchterlonie /

Richard C. Auchterlonie
Reg. No. 30,607

NOVAK DRUCE & QUIGG, LLP
1000 Louisiana, 53rd Floor
Houston, TX 77002
713-571-3460

VIII. CLAIMS APPENDIX

8. A method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, said method comprising:

the snapshot copy facility receiving a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies; and

the snapshot copy facility responding to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies;

wherein the snapshot copy facility has an index for each snapshot copy for indicating changes between said each snapshot copy and a next snapshot copy of the production file system, and the method includes scanning the index for the specified older one of the snapshot copies,

which includes scanning the indices for a sequence of the snapshot copies including the index for the specified older one of the snapshot copies and a respective index for each of a plurality of snapshot copies of the production file system that are both younger than the specified older one snapshot copies and older than the specified younger one of the snapshot copies, and

wherein the indices for the sequence of the snapshot copies are scanned by a program routine having an outer loop indexing blocks of data in the file system, and an inner loop indexing the snapshot copies in the sequence of the snapshot copies.

9. A method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, said method comprising:

the snapshot copy facility receiving a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies; and

the snapshot copy facility responding to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies;

wherein the snapshot copy facility has an index for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy, and the method includes scanning the index for the specified younger one of the snapshot copies, and when the index indicates that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

16. A method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, the snapshot copy facility having an index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system, wherein the method comprises:

scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies;

wherein the indices for the sequence of the snapshot copies are scanned by a program routine having an outer loop indexing respective blocks, and an inner loop indexing snapshot copies in the sequence of the snapshot copies; and

wherein the snapshot copy facility has a meta bit map for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy, and the method includes scanning the meta bit map for the specified younger one of the snapshot copies, and when the meta bit map is found to indicate that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies by scanning the indices for the sequence of the snapshot copies.

17. A method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, the snapshot copy facility having a first index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system

and that have a "before image" saved for said each snapshot copy, the snapshot copy facility having a second index for said each snapshot copy for indicating blocks of data that are not in use in said each snapshot copy; said method comprising:

responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by accessing the second index for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices for a sequence of the snapshot copies to determine blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

18. The method as claimed in claim 17, which also includes accessing at least one of the second indices for the snapshot copies in the sequence of the snapshot copies and finding that at least one of the blocks is not in use in at least one of the snapshot copies in the sequence of the snapshot copies to determine that said at least one of the blocks has changed between the older one of the snapshot copies and the younger one of the snapshot copies not changed.

33. A snapshot copy facility comprising:

storage for storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time; and

at least one processor programmed for receiving a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies; and for responding to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies;

wherein the snapshot copy facility has an index for each snapshot copy for indicating changes between said each snapshot copy and a next snapshot copy of the production file system, and said at least one processor is programmed for scanning the index for the specified older one of the snapshot copies,

wherein said at least one processor is programmed for scanning the indices for a sequence of the snapshot copies including the index for the specified older one of the snapshot copies and a respective index for each of a plurality of snapshot copies of the production file system that are both younger than the specified older one snapshot copies and older than the specified younger one of the snapshot copies, and

wherein said at least one processor is programmed for scanning the indices for the sequence of the snapshot copies by a program routine having an outer loop indexing the blocks, and an inner loop indexing the snapshot copies in the sequence of the snapshot copies.

34. A snapshot copy facility comprising:

storage for storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time; and

at least one processor programmed for receiving a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies; and for responding to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies;

wherein the snapshot copy facility has an index for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy, and said at least one processor is programmed for scanning the index for the specified younger one of the snapshot copies, and when the index indicates that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

41. A snapshot copy facility comprising:

storage for storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time;

an index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system, and

at least one processor programmed for scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies,

which includes a meta bit map for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy, and wherein said at least one processor is programmed for scanning the meta bit map for the specified younger one of the snapshot copies, and when the meta bit map is found to indicate that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies by scanning the indices for the sequence of the snapshot copies.

42. A snapshot copy facility comprising:

storage for storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time;

a first index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system and that have a "before image" for said each snapshot copy stored in the storage,

a second index for each snapshot copy for indicating blocks of data that are not in use in said each snapshot copy, and

at least one processor programmed for responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by accessing the second index for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices for a sequence of the snapshot copies to determine blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

43. The snapshot copy facility as claimed in claim 42, wherein said at least one processor is also programmed for accessing at least one of the second indices for the snapshot copies in the sequence of the snapshot copies and finding that at least one of the blocks is not in use in at least one of the snapshot copies in the sequence of the snapshot copies to determine that said at least one of the blocks has changed between the older one of the snapshot copies and the younger one of the snapshot copies not changed.

54. A program storage device containing a program for a snapshot copy facility, the snapshot copy facility storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, wherein the program is executable for responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies,

wherein the snapshot copy facility has an index for each snapshot copy for indicating changes between said each snapshot copy and a next snapshot copy of the production file system, and the program is executable for scanning the index for the specified older one of the snapshot copies,

wherein the program is executable for scanning the indices for a sequence of the snapshot copies including the index for the specified older one of the snapshot copies and a respective index for each of a plurality of snapshot copies of the production file system that are both younger than the specified older one snapshot copies and older than the specified younger one of the snapshot copies,

wherein the program is executable for scanning the indices for a sequence of the snapshot copies including the index for the specified older one of the snapshot copies and a respective index for each of a plurality of snapshot copies of the production file system that are both

younger than the specified older one snapshot copies and older than the specified younger one of the snapshot copies, and

wherein the program is executable for scanning the indices for the sequence of the snapshot copies by a program routine having an outer loop indexing the blocks, and an inner loop indexing the snapshot copies in the sequence of the snapshot copies.

59. A program storage device containing a program for a snapshot copy facility, the snapshot copy facility having a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, and an index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system, wherein the program is executable for scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies,

wherein the snapshot copy facility has a meta bit map for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy, and wherein the program storage device is executable for scanning the meta bit map for the specified younger one of the snapshot copies, and when the meta bit map is found to indicate that a block is not known to be invalid, then determining whether the block has changed between the specified

older one of the snapshot copies and the specified younger one of the snapshot copies by scanning the indices for the sequence of the snapshot copies.

60. A program storage device containing a program for a snapshot copy facility, the snapshot copy facility having a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, a first index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system and that have a "before image" for said each snapshot copy stored in the snapshot copy facility, and a second index for each snapshot copy for indicating blocks of data that are not in use in said each snapshot copy, wherein the program is executable for responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by accessing the second index for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices for a sequence of the snapshot copies to determine blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

61. The program storage device as claimed in claim 60, wherein the program is executable for accessing at least one of the second indices for the snapshot copies in the sequence of the snapshot copies and finding that at least one of the blocks is not in use in at least one of the snapshot copies in the sequence of the snapshot copies to determine that said at least one of the blocks has changed between the older one of the snapshot copies and the younger one of the snapshot copies not changed.

IX. EVIDENCE APPENDIX

None.

X. RELATED PROCEEDINGS APPENDIX

None.

UNITED STATES PATENT AND TRADEMARK OFFICE

T-D

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/603,411	06/25/2003	Philippe Armangau	EMCR:0095NPU	4172

27927 7590 01/10/2007

RICHARD AUCHTERLONIE
NOVAK DRUCE & QUIGG, LLP
1000 LOUISIANA
53RD FLOOR
HOUSTON, TX 77002

EXAMINER	
ONI, OLUBUSOLA	
ART UNIT	**PAPER NUMBER**
2168	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	01/10/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Office Action Summary	Application No.	Applicant(s)	
	10/603,411	ARMANGAU ET AL.	
	Examiner	Art Unit	
	OLUBUSOLA ONI	2168	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *31 October 2006*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *8,9,16-18,33,34,41-43,54 and 59-61* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *8,9, 16-18, 33-34, 41-43, 54, 59, 60, 61* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

Reopening Prosecution After Appeal

1. ¶ 12.187 Reopening of Prosecution After Appeal Brief or Reply Brief

In view of the appeal brief filed on 08/16/2006, PROSECUTION IS HEREBY REOPENED. In the final office action, claim 9 was considered and it was rejected. However, examiner mistreated the claimed limitations. Thus claims 17-48, 42-43 and 60-61 were not addressed in the office action. Therefore, it would be appropriate to reopen the prosecution to correct those errors above. All Claims rejected as the same grounds of rejection from the previous office action. To avoid abandonment of the application, appellant must exercise one of the following two options:

(1) file a reply under 37 CFR 1.111 (if this Office action is non-final) or a reply under 37 CFR 1.113 (if this Office action is final); or,

(2) initiate a new appeal by filing a notice of appeal under 37 CFR 41.31 followed by an appeal brief under 37 CFR 41.37. The previously paid notice of appeal fee and appeal brief fee can be applied to the new appeal. If, however, the appeal fees set forth in 37 CFR 41.20 have been increased since they were previously paid, then appellant must pay the difference between the increased fees and the amount previously paid.

A Supervisory Patent Examiner (SPE) has approved of reopening prosecution by signing below:

Claim Rejections - 35 USC § 102

2. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

3. Claims 8,33 and 54 are rejected under 35 U.S.C. 102(e) as being anticipated by Goldstein et al. (Pub No. 2004/0163009) hereinafter "Goldenstein"

For claim 8, Goldenstein teaches " A method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time"(See paragraph [0029])

"the snapshot copy facility receiving a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies"

(See paragraph [0011] and [0024] wherein Goldstein's teaches include acquiring the difference between the base state snapshot and the data base volume snapshot as implied in applicant's claim language)

"the snapshot copy facility responding to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies"(See paragraph [0030-0032] wherein Goldstein's teachings include the difference between the base state snapshot and the data base volume snapshot, as implied in applicant's claim language)

"wherein the snapshot copy facility has an index for each snapshot copy for indicating changes between said each snapshot copy and a next snapshot copy of the production file system, and the method includes scanning the index for the specified older one of the snapshot copies"(See paragraph [0029] and [0043] and fig. 10 of Goldstein's drawings illustrates the repetitive obtaining of a snapshot difference list, as implied in applicant's claim language)

"which includes scanning the indices for a sequence of the snapshot copies including the index for the specified older one of the snapshot copies and a respective index for each of a plurality of snapshot copies of the production file system that are both younger than the specified older one snapshot copies and older than the specified younger one of the snapshot copies"(See paragraph [0027-0029] and fig. 3 Goldstein's drawings illustrates the base state snapshot and a subsequent series of data volume snapshots).

"wherein the indices for the sequence of the snapshot copies are scanned by a program routine having an outer loop indexing blocks of data in the file system, and an inner loop

indexing the snapshot copies in the sequence of the snapshot copies" (See paragraph [0011], [0024-0033], [00421-0043] and fig 3, 4, 6& 7 wherein Goldstein's teachings involve the determination of blocks that have changed, and also include indexing snapshot copies, thus teaches are synonymous).

As per claim 33, Goldstein teaches "storage for storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time"(See paragraph [0024] and [0029]) "and at least one processor programmed for receiving a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies" (See paragraphs [0011] and [0024] wherein Goldstein's teachings include acquiring the difference between the base state snapshot and the data base volume snapshot as implied in applicant's claim language) "and for responding to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies" (See paragraphs [0030] and [0032] wherein Goldstein's teachings include the difference between the base state snapshot and the data base volume snapshot, as implied in applicant's claim language). "wherein the snapshot copy facility has an index for each snapshot copy for indicating changes between said each snapshot copy and a next snapshot copy of the production file system, and said at least one processor is programmed for scanning the index for the specified older one of the snapshot copies" (See paragraphs [0029] and

[0043] and Fig 10 of Goldstein's drawings illustrate the repetitive obtaining of a snapshot difference list, as implied in applicants claim language).

"wherein said at least one processor is programmed for scanning the indices for a sequence of the snapshot copies including the index for the specified older one of the snapshot copies and a respective index for each of a plurality of snapshot copies of the production file system that are both younger than the specified older one snapshot copies and older than the specified younger one of the snapshot copies" (See paragraphs [0027] and [0028] and Fig 3 of Goldstein's drawings illustrates the base state snapshot and a subsequent series of data volume snapshots).

"wherein said at least one processor is programmed for scanning the indices for the sequence of the snapshot copies by a program routine having an outer loop indexing the blocks, and an inner loop indexing the snapshot copies in the sequence of the snapshot copies" (See paragraph [0011], [0024-0033], [00421-0043] and fig 3, 4, 6& 7 wherein Goldstein's teachings involve the determination of blocks that have changed, and also include indexing snapshot copies, thus teaches are synonymous).

As per claim 54,Goldstein teaches "a program storage device containing a program for a snapshot copy facility the snapshot copy facility storing a plurality of snapshot copies of a production file system"(See paragraph [0024] and [0029]),

"each of the snapshot copies being a prior state of the production file system at a respective point in time, wherein the program is executable for responding to a request for the difference between a specified older one of the snapshot copies and a specified

younger one of the snapshot copies by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies"(See paragraphs [0011] and [0024] and [0027] and [0029] wherein Goldstein's teachings included identifying and producing a list of data blocks which differ between the subsequent snapshots, thus teachings are synonymous).

"wherein the snapshot copy facilities has an index for each snap shot copy for indicating changes between said each snapshot copy and a next snapshot copy of the production file system, and the program is executable for scanning the index for the specified older one of the snapshot copies" (See paragraphs [0029] and [0043] and Fig 10 of Goldstein's drawings illustrate the repetitive obtaining of a snapshot difference list, as implied in applicants claim language).

"wherein the program is executable for scanning the indices for a sequence of the snapshot copies including the index for the specified older one of the snapshot copies and a respective index for each of a plurality of snapshot copies of the production file system that are both younger than the specified older one snapshot copies and older than the specified younger one of the snapshot copies" (See paragraphs [0027] and [0028] and Fig 3 of Goldstein's drawings illustrates the base state snapshot and a subsequent series of data volume snapshots).

"wherein the program is executable for scanning the indices for a sequence of the snapshot copies including the index for the specified older one of the snapshot copies and a respective index for each of a plurality of snapshot copies of the production file system that are both younger than the specified older one snapshot copies and older

than the specified younger one of the snapshot copies" (See paragraphs [0027] and [0028] and Fig 3 of Goldstein's drawings illustrates the base state snapshot and a subsequent series of data volume snapshots).

"wherein the program is executable for scanning the indices for the sequence of the snapshot copies by a program routine having outer loop indexing the blocks, and an inner loop indexing the snapshot copies in the sequence of the snapshot copies" (See paragraph [0011], [0024-0033], [00421-0043] and fig 3, 4, 6& 7 wherein Goldstein's teachings involve the determination of blocks that have changed, and also include indexing snapshot copies, thus teaches are synonymous).

4. Claims 9,16-18, 34, 41-43, 59, 60 and 61 are rejected under 35 U.S.C. 103(a) as being unpatentable over Goldstein in view of Ohran et al (Pub No. US 20020112134) (hereinafter Ohran).

As per claim 9, Goldstein teaches "a method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time"(See paragraph [0024] and [0029]) "said method comprising:

"the snapshot copy facility receiving a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies" (See paragraphs [0030] and [0032] wherein Goldstein's teachings include the difference between the base state snapshot and the data base volume snapshot, as implied in

applicant's claim language)"and the snapshot copy facility responding to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies" (See paragraphs [0024]-[0029] and Fig 3&4).

Goldstein teaches, "determining whether there has been a change between the specified older one of the snapshot... of the snapshot copies" (See paragraphs [0024]-[0028] and Fig 3 of Goldstein's drawings illustrates comparing the base state snapshot and a subsequent series of data volume snapshots. Wherein valid data volume is produced at a consistent state and comparing the based state snapshot and a subsequent series of data volume snapshots to confirm changes, which can only be done if block is valid).

Goldstein does not teach "wherein the snapshot copy facility has an index for each snapshot copy for indicating blocks of data that are know to be invalid in said each snapshot copy..."

However, Ohran's teachings and drawings illustrate writing invalid or corrupted data to certain data blocks in the mass storage device (See paragraph [0015], [0045] and [Fig 3])

Therefore, it would have been obvious at the time of the invention for one of ordinary skill in the art to have modify Goldstein by teachings of Ohran, wherein Ohran's teaches of separating invalid data, while a valid set of data is eventually used to reconstruct the invalid data, and been combined with Goldstein's method will enhance checking for changes of valid data.

As per claim 16, this claim is rejected on the grounds corresponding to the argument given above for rejecting claim 9 above including the following reasons:

"A method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, the snapshot copy facility having an index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system" See paragraphs [0011] and [0024] and [0027] and [0029] wherein Goldstein's teachings included identifying and producing a list of data blocks which differ between the subsequent snapshots, thus teachings are synonymous).

"scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies" (See paragraphs [0027] and [0028] and Fig 3 of Goldstein's drawings illustrates the base state snapshot and a subsequent series of data volume snapshots).

"wherein the indices for the sequence of the snapshot copies are scanned by a program routine having an outer loop indexing respective blocks, and an inner loop indexing snapshot copies in the sequence of the snapshot copies" (See paragraph [0011], [0024-

0033], [00421-0043] and fig 3, 4, 6& 7 wherein Goldstein's teachings involve the determination of blocks that have changed, and also include indexing snapshot copies, thus teaches are synonymous).

Goldstein teaches, "determining whether there has been a change between the specified older one of the snapshot..." (See paragraphs [0024]-[0029] and Fig 3 of Goldstein's drawings illustrates comparing the base state snapshot and a subsequent series of data volume snapshots. Wherein valid data volume is produced at a consistent state and comparing the based state snapshot and a subsequent series of data volume snapshots to confirm changes, which can only be done if block is valid).

Goldstein does not teach "wherein the snapshot copy facility has a meta bit map for each snapshot copy for indicating blocks of data that are know to be invalid in ...". However, Ohran's teachings and drawings illustrate writing invalid or corrupted data to certain data blocks in the mass storage device (See paragraph [0015] and [Fig 3]) Therefore, it would have been obvious at the time of the invention for one of ordinary skill in the art to have modify Goldstein by teachings of Ohran, wherein Ohran's teaches of separating invalid data, while a valid set of data is eventually used to reconstruct the invalid data, and been combined with Goldstein's method will enhance checking for changes of valid data.

For claim 34, this claim is rejected on grounds corresponding to the arguments given above for rejected claim 9 and is similarly rejected.

As per claim 41 this claim is rejected on the grounds corresponding to the argument given above for rejecting claims 16 above, including the following reasons:
Goldstein teaches "storage for storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time"(See paragraph [0024] and [0029])
"an index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system" (See paragraphs [0011] and [0024] and [0027] and [0029] wherein Goldstein's teachings included identifying and producing a list of data blocks which differ between the subsequent snapshots, thus teachings are synonymous).
"at least one processor programmed for scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies" (See paragraphs [0027] and [0028] and Fig 3 of Goldstein's drawings illustrates the base state snapshot and a subsequent series of data volume snapshots).

As per claim 59, this claim is rejected on the grounds corresponding to the argument given above for rejecting claim 16 above including the following reasons:
Goldstein teaches "a program storage device containing a program for a snapshot copy facility, the snapshot copy facility having a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, and an index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system" (See paragraphs [0011] and [0024] and [0027] and [0029] wherein Goldstein's teachings included identifying and producing a list of data blocks which differ between the subsequent snapshots, thus teachings are synonymous).
"wherein the program is executable for scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies" (See paragraphs [0027] and [0028] and Fig 3 of Goldstein's drawings illustrates the base state snapshot and a subsequent series of data volume snapshots).

As per claim 17, 18, 42, 43, 60and 61 Goldstein teaches "A method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file

system, each of the snapshot copies being a prior state of the production file system at a respective point in time"(See paragraph [0029]) the snapshot copy facility having a first index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy ..." (See paragraphs [0029] and [0043] and Fig 10)

Goldstein teaches "responding to a request for the difference between a specific older one of the snapshot copies and a specified younger one of the snapshot copies...snapshot copies and older than the younger one of the snapshot copies". (See paragraph [0030-0032] wherein Goldstein's teachings include the difference between the base state snapshot and the data base volume snapshot, as implied in applicant's claim language).

Goldstein does not teach "the snapshot copy facility having a second index for each snapshot copy for indicating blocks of data that are not in use" Ohran teachings include writing invalid or corrupted data to a certain data block. Wherein the abstract states data loss could be caused by data blocks becoming corrupt or lost, therefore data not in use is equivalent to data loss. (See paragraph [0015] and Fig 3)

It would have been obvious at the time of the invention for one of ordinary skill in the art to have modified Goldstein by the teachings of Ohran, to access the first and second index before checking for changes between the snapshot copies.

Response to Amendment

5. Applicant's arguments filled March 23, 2006 have been fully considered but they are not persuasive. The examiner respectfully traverses applicant's argument.

As per claims 8, 33 and 54 applicant's argued Goldstein does not explicitly teaches "a plurality of snapshot copies of the production file..." On the contrary at paragraph [0027-0029] and Fig 3 of Goldstein's drawings illustrates the comparison of the base state snapshot and a subsequent series (plurality) of data volume snapshots. The comparison is done based on the time, wherein the first state snapshot is compared to the second state snapshot, which is later been compared to the third state snapshot, as implied in applicant's claim language). Applicant's argued Goldstein does not teach "scanning procedure including inner and outer loops that determine the blocks that have changed over a series of at least three successive snapshots". On the contrary Goldstein's teaches at paragraph [0011], [0024-0033], [00421-0043] and fig 3, 4, 6& 7. Wherein Goldstein's teachings involve acquiring snapshots of consistent states of data volume, wherein the snapshots are compared to produce a list of blocks that have changed between the snapshots, i.e., the determination of blocks that have changed, thus teaches are synonymous).

As per claim 9 and 34, applicant argued against the references individually however, one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller,* 642 F.2d 413, 208 USPQ 871 (CCPA 1981); *In re Merck & Co.,* 800 F.2d 1091, 231 USPQ 375 (Fed. Cir.

1986). Applicant argued that Goldstein does not teach "determining a list of snapshot differences requires a determination that the blocks are valid or not in a younger snapshot".

On the contrary Goldstein teaches, "determining whether there has been a change between the specified older one of the snapshot... of the snapshot copies" (See paragraphs [0024]-[0028] and Fig 3 of Goldstein's drawings illustrates comparing the base state snapshot and a subsequent series of data volume snapshots. Wherein valid data volume is produced at a consistent state and comparing the based state snapshot and a subsequent series of data volume snapshots to confirm changes, which can only be done if block is valid).

However, Goldstein does not teach "wherein the snapshot copy facility has an index for each snapshot copy for indicating blocks of data that are know to be invalid in said each snapshot copy..."

However, Ohran's teachings and drawings illustrate writing invalid or corrupted data to certain data blocks in the mass storage device (See paragraph [0015], [0045] and [Fig 3])

Therefore, it would have been obvious at the time of the invention for one of ordinary skill in the art to have modify Goldstein by teachings of Ohran, wherein Ohran's teaches of separating invalid data, while a valid set of data is eventually used to reconstruct the invalid data, and been combined with Goldstein's method will enhance checking for changes of valid data.

Applicant also argued that Ohran does not teach "checking for blocks not know to be valid before checking for changes".

On the contrary, Ohran's teachings at paragraph 0015 and drawings at Fig 3 illustrate writing invalid or corrupted data to certain data blocks in the mass storage device. Also at paragraph 0045 of Ohran's teachings include the determining whether there has been a change between the data sets after confirming the valid data block.

As per claim 16, 41 and 59 applicant argued that Goldstein does not teach "the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies" , "wherein the indices for the sequence of the snapshot copies are scanned by a program routine having an outer loop indexing respective blocks, and an inner loop indexing snapshot copies in the sequence of the snapshot copies".

On the contrary Goldstein teaches at paragraph 0027, 0028, 0029 and Fig. 3 "the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies"(wherein Goldstein's drawings also illustrates the base state snapshot and a subsequent series of data volume snapshots).

Goldstein also teaches at 0011, 0024-0033, 0042-0043 and fig 3, 4, 6& 7 "the indices for the sequence of the snapshot copies are scanned by a program routine having an

outer loop indexing respective blocks, and an inner loop indexing snapshot copies in the sequence of the snapshot copies" (Wherein Goldstein's teachings involve the determination of blocks that have changed, and also include indexing snapshot copies, thus teaches are synonymous).

As per claims 17,42 and 60 applicant's argued Ohran does not teach "writing invalid or corrupted data to a certain data block" wherein examiner stated that the abstract teaches data loss could be caused by data block becoming corrupt or lost, therefore data not in use is equivalent to data loss (See paragraph [0015] and fig. 3]) as stated by examiner. Applicant's argued corrupt data or data lost is not equivalent to data not in use, but if there is no need to reconstruct data not in use as stated by applicant's on pg 37 remarks/argument, then data not in use can be equivalent to data lost.

CONCLUSION

5. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to OLUBUSOLA ONI whose telephone number is 571-272-2738. The examiner can normally be reached on 10.00-6.30PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, TIM VO can be reached on 571-272-3642. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

OLUBUSOLA ONI KP
Examiner
Art Unit 2168

TIM VO
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

EXPRESS MAIL MAILING LABEL NUMBER EV 318623698 US

PATENT

10830.0095.NPUS00

APPLICATION FOR UNITED STATES LETTERS PATENT

for

REPLICATION OF SNAPSHOT USING A FILE SYSTEM COPY DIFFERENTIAL

By

Philippe Armangau
Milena Bergant

BACKGROUND OF THE INVENTION

1. Limited Copyright Waiver

A portion of the disclosure of this patent document contains computer code listings and command formats to which the claim of copyright protection is made. The copyright owner has no objection to the facsimile reproduction by any person of the patent document or the patent disclosure, as it appears in the U.S. Patent and Trademark Office patent file or records, but reserves all other rights whatsoever.

2. Field of the Invention

The present invention relates generally to data storage systems, and more particularly to a snapshot copy facility for a data storage system.

3. Description of the Related Art

Snapshot copies of a file system have been used for a variety of data processing and storage management functions such as storage backup, transaction processing, and software debugging. A snapshot copy facility, for example, stores a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility has a bit map, a block map, and a save volume for each snapshot copy. The bit map for each snapshot copy indicates blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system. The save volume for each snapshot copy stores the "before images" of the blocks that have changed between the snapshot copy and the next snapshot copy of the production file system. The block map for each snapshot copy provides the save volume block address given the block address of a block that has changed between the snapshot copy and the

next snapshot of the production file system.

SUMMARY OF THE INVENTION

In accordance with a first aspect of the invention, there is provided a method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility receives a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies. The snapshot copy facility responds to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

In accordance with another aspect, the invention provides a method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility has an index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system. The method includes scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies. The sequence of the snapshot copies includes the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

In accordance with another aspect, the invention provides a method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility has a first index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system and that have a "before image" saved for said each snapshot copy. The snapshot copy facility has a second index for each snapshot copy for indicating blocks of data that are not in use in the snapshot copy. The method includes responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by accessing the second index for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices for a sequence of the snapshot copies to determine blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies. The sequence of the snapshot copies includes the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

In accordance with yet another aspect, the invention provides a method of operating a network file server. The network file server has a snapshot copy facility for storing a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The

network file server receives a request for an update to a specified snapshot copy of the production file system. The network file server responds to the request by checking whether the snapshot copy facility contains the specified snapshot copy of the production file system, and upon finding that the snapshot copy facility contains the specified snapshot copy of the production file system, the network file server returns the difference between the specified snapshot copy of the production file system and a more recent snapshot copy of the production file system.

In accordance with yet another aspect, the invention provides a method of operation in a data processing network having a client and a network file server. The network file server stores a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The client has a local version of an older one of the snapshot copies. The method provides the client with a younger one of the snapshot copies. The method includes the network file server determining the difference between the younger one of the snapshot copies and the older one of the snapshot copies. The network file server transmits the difference between the younger one of the snapshot copies and the older one of the snapshot copies to the local version of the older one of the snapshot copies. The difference between the younger one of the snapshot copies and the older one of the snapshot copies is written into the local version of the older one of the snapshot copies to produce a local version of the younger one of the snapshot copies.

In accordance with another aspect, the invention provides a snapshot copy facility. The snapshot copy facility includes storage for storing a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the

production file system at a respective point in time. The snapshot copy facility also includes at least one processor programmed for receiving a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies; and for responding to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

In accordance with yet another aspect, the invention provides a snapshot copy facility. The snapshot copy facility includes storage for storing a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility includes an index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system. The snapshot copy facility also includes at least one processor programmed for scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

In accordance with another aspect, the invention provides a snapshot copy facility. The snapshot copy facility includes storage for storing a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility has a first index for each snapshot copy for indicating blocks of data in the production file system

that have changed between the snapshot copy and a next snapshot copy of the production file system and that have a "before image" for the snapshot copy stored in the storage. The snapshot copy facility has a second index for each snapshot copy for indicating blocks of data that are not in use in the snapshot copy. The snapshot copy facility also has at least one processor programmed for responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by accessing the second index for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices for a sequence of the snapshot copies to determine blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies. The sequence of the snapshot copies includes the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

In accordance with still another aspect, the invention provides a network file server including a snapshot copy facility for storing a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The network file server is programmed for receiving a request for an update to a specified snapshot copy of the production file system, and responding to the request by checking whether the snapshot copy facility contains the specified snapshot copy of the production file system, and upon finding that the snapshot copy facility contains the specified snapshot copy of the production file system, returning

the difference between the specified snapshot copy of the production file system and a more recent snapshot copy of the production file system.

In accordance with another aspect, the invention provides a program storage device containing a program for a snapshot copy facility. The snapshot copy facility stores a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The program is executable for responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

In accordance with yet another aspect, the invention provides a program storage device containing a program for a snapshot copy facility. The snapshot copy facility has a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility also has an index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system. The program is executable for scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

In accordance with another aspect, the invention provides a program storage device containing a program for a snapshot copy facility. The snapshot copy facility has a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility has a first index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system and that have a "before image" for the snapshot copy stored in the snapshot copy facility. The snapshot copy facility has a second index for each snapshot copy for indicating blocks of data that are not in use in the snapshot copy. The program is executable for responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by accessing the second index for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices for a sequence of the snapshot copies to determine blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies. The sequence of the snapshot copies includes the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

In accordance with a final aspect, the invention provides a program storage device containing a program for a network file server. The network file server includes a snapshot copy facility for storing a plurality of snapshot copies of a production file

system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The program is executable for receiving a request for an update to a specified snapshot copy of the production file system, and responding to the request by checking whether the snapshot copy facility contains the specified snapshot copy of the production file system, and upon finding that the snapshot copy facility contains the specified snapshot copy of the production file system, returning the difference between the specified snapshot copy of the production file system and a more recent snapshot copy of the production file system.

BRIEF DESCRIPTION OF THE DRAWINGS

Other objects and advantages of the invention will become apparent upon reading the following detailed description with reference to the accompanying drawings wherein:

FIG. 1 is a block diagram of an IP network including multiple hosts and multiple data mover computers;

FIG. 2 is a block diagram showing a primary data mover distributing remote copy data to multiple secondary data movers in the IP network by establishing a Transmission Control Protocol (TCP) connection with each of the secondary data movers;

FIG. 3 is a block diagram showing a primary data mover distributing remote copy data to multiple data movers through forwarder data movers;

FIG. 4 is a block diagram showing a shared save volume used to buffer local copy data transmitted from a primary data mover to a secondary data mover;

FIG. 5 is a block diagram showing a primary save volume and a secondary save volume;

FIG. 6 is a flowchart showing local replication in the system of FIG. 4;

FIG. 7 is a flowchart showing remote replication in the system of FIG. 5;

FIG. 8 is a block diagram of a primary site, including layered programming in a primary data mover;

FIG. 9 is a block diagram of a secondary site, including layered programming in a secondary data mover;

FIG. 10 is a flowchart of a process of replication at the primary site of FIG. 8;

FIG. 11 is a flowchart of a procedure for producing a new remote copy of a primary file system concurrent with ongoing replication and multicasting of modifications to the primary file system;

FIG. 12 is a flowchart of an IP-replication send-thread introduced in FIG. 8;

FIG. 13 is a block diagram of a volume multicast level in the data mover programming of FIG. 8 and FIG. 9;

FIG. 14 is a block diagram of the RCP level in the primary data mover programming of FIG. 8;

FIG. 15 is a block diagram of the RCP level in the secondary data mover programming of FIG. 9;

FIG. 16 is a block diagram of a RPC forwarder at the RPC level in a forwarder data mover;

FIG. 17 is a flowchart of an inbound RCP session in the secondary data mover;

FIG. 18 is a block diagram showing a forwarder data mover performing local replication;

FIG. 19 is a block diagram showing the sharing of a data mover's single TCP port for RCP connections with Hypertext Transfer Protocol (HTTP) connections;

FIG. 20 is a block diagram showing the state of a remote replication system before a disaster at the primary site;

FIG. 21 is a flowchart of a failover and recovery method performed upon the remote replication system of FIG. 20 in order to recover from the disaster at the primary site;

FIG. 22 is a block diagram showing the state of the remote replication system of FIG. 20 after failover from the primary site to the secondary site;

FIG. 23 is a flowchart of a subroutine of failover with a checkpoint and without sync, as used in the flowchart of FIG. 21;

FIG. 24 shows a block map that can be used to create a snapshot copy of the secondary file system at a restart point during the failover of FIG. 23;

FIG. 25 shows a snapshot copy facility that can be used to create the snapshot copy of the secondary file system at the restart point during the failover of FIG. 23;

FIG. 26 is a flowchart of a procedure for writing a block of data to a production file system in the snapshot copy facility of FIG. 25;

FIG. 27 is a flowchart of a procedure for reading a block of data from a snapshot file system in the snapshot copy facility of FIG. 25;

FIG. 28 is a flowchart of a subroutine to resync the primary file system with the secondary file system, as used in the flowchart of FIG. 21;

FIG. 29 shows the state of the remote replication system of FIG. 20 during the resync procedure;

FIG. 30 is a flowchart of a subroutine to failback to the primary file system, as used in the flowchart of FIG. 21;

FIG. 31 is a flowchart of execution of a failover command having a sync option and a checkpoint option;

FIG. 32 is a flowchart of a subroutine for failover without sync, as used in the flowchart of FIG. 31;

FIG. 33 is a subroutine for failover with sync, as used in the flowchart of FIG. 31;

FIGS. 34 and 35 comprise a flowchart of a procedure for copying snapshot delta for snapshots L to M;

FIGS. 36 shows a block diagram of a data network in which snapshot deltas are transmitted over a wide-area network from a network file server to a local file server in order to update the local file system as needed;

FIGS. 37 and 38 comprise a flowchart of a procedure for replicating snapshots in the data network of FIG. 36;

FIG. 39 is a modified version of the flowchart of FIG. 34, showing how to copy a snapshot delta from a snapshot copy facility that keeps track of blocks of a production file system that are not in use in the snapshot copies of the production file system;

FIG. 40 shows a specific instance of the flowchart of FIG. 39 for a snapshot copy facility that uses respective meta bit maps for indicating the blocks in a production file system that are not used in the snapshot copies of the production file system;

FIG. 41 is a flowchart for writing a specified data block to the production file system for a snapshot copy facility that uses a meta bit map for indicating the blocks in a production file system that are not presently in use;

FIG. 42 is a diagram of a preferred organization of multiple snapshots in the snapshot copy facility;

FIG. 43 shows a bit map including a page table and a set of pages, for use in the snapshot organization of FIG. 42;

FIG. 44 shows a block map including a hash table and hash lists, for use in the snapshot organization of FIG. 42;

FIG. 45 shows a specific construction for and interpretation of a meta bit map for the production volume; and

FIG. 46 shows an alternative interpretation of a meta bit map for the production volume.

While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof have been shown by way of example in the drawings and will be described in detail. It should be understood, however, that it is not intended to limit the form of the invention to the particular forms shown, but on the contrary, the intention is to cover all modifications, equivalents, and alternatives falling within the scope of the invention as defined by the appended claims.

DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

The present invention was developed to facilitate data recovery in a data network using a remote data replication facility. Therefore, the following detailed description will first describe the remote data replication facility, and will then describe data recovery for the remote data replication facility.

Remote Data Replication Facility

FIG. 1 shows an IP network 220 including multiple network file servers 221, 222, and multiple hosts 223, 224, 225. The hosts and network file servers, for example, can be distributed world wide and linked via the Internet. Each of the network file servers 221, 222, for example, has multiple data movers 226, 227, 228, 232, 233, 234, for moving data between the IP network 220 and the cached disk array 229, 235, and a control station 230, 236 connected via a dedicated dual-redundant data link 231, 237 among the data movers for configuring the data movers and the cached disk array 229, 235. Further details regarding the network file servers 221, 222 are found in Vahalia et al., U.S. Patent 5,893,140, incorporated herein by reference.

In operation, it is desired for each of the network file servers 221, 222 to provide read-only access to a copy of the same file system. For example, each of the network file servers could be programmed to respond to user requests to access the same Internet site. The IP network 220 routes user requests to the network file servers 221, 222 in the same continent or geographic region as the user. In this fashion, the user load is shared among the network file servers.

In the wide-area network of FIG. 1, it is desired to perform read-write updating of the respective file system copies in the network file servers 221, 222 while permitting concurrent read-only access by the hosts. It is also desired to distribute the updates over the IP network.

There are a number of ways that updates could be distributed over the IP network from a primary data mover to multiple secondary data movers. As shown in FIG. 2, for example, a primary data mover establishes a connection 242, 243, 244 in accordance with

the industry-standard Transmission Control Protocol (TCP) over the IP network 220 to each secondary data mover 245, 246, 247, and then concurrently sends the updates to each secondary data mover over the TCP connection. When the updates need to be distributed to a large number of secondary data movers, however, the amount of time for distributing the updates may become excessive due to limited resources (CPU execution cycles, connection state, or bandwidth) of the primary data mover 241. One way of extending these limited resources would be to use existing IP routers and switches to implement "fan out" from the primary data mover 241 to the secondary data movers 245, 246, 247. Still, a mechanism for reliability should be layered over the Internet Protocol.

FIG. 3 shows that the time for distributing updates from a primary data mover 251 to a large number of secondary data movers 254, 255, 256, 257 can be reduced by using intermediate data movers 252, 253 as forwarders. The primary data mover 251 sends the updates to the forwarder data movers 252, 253, and each of the forwarder data movers sends the updates to a respective number of secondary data movers. The forwarder data movers 252, 253 may themselves be secondary data movers; in other words, each may apply the updates to its own copy of the replicated read-only file system. The distribution from the primary data mover 251 to the forwarder data movers 252, 253 can be done in a fashion suitable for wide-area distribution (such as over TCP connections). The forwarding method of replication of FIG. 3 also has the advantage that the distribution from each forwarder data mover to its respective data movers can be done in a different way most suitable for a local area or region of the network. For example, some of the forwarder data movers could use TCP connections, and others could use a combination of

TCP connections for control and UDP for data transmission, and still other forwarders could be connected to their secondary data movers by a dedicated local area network.

For implementing the replication method of FIG. 3 over the Internet Protocol, there are a number of desired attributes. It is desired to maintain independence between the primary data mover and each of the secondary data movers. For example, a new secondary data mover can be added at any time to replicate an additional remote copy. The primary data mover should continue to function even if a secondary data mover becomes inoperative. It is also desired to maintain independence between the replication method and the IP transport method. Replication should continue to run even if the IP transport is temporarily inactive. It is desired to recover in a consistent fashion from a panic or shutdown and reboot. A record or log of the progress of the replication can be stored for recovery after an interruption. It is desired to build re-usable program blocks for the replication function, so that the program blocks for the replication function can be used independent of the location of the primary file system or its replicas.

In a preferred implementation, independence between the replication process, the IP transport method, and the primary file system being replicated, is ensured by use of a save volume. The save volume is a buffer between the data producer (i.e., the host or application updating the primary file system), the replication process, and the data consumer (the secondary data movers). The save volume stores the progress of the replication over the Internet Protocol so as to maintain the consistency of the replication process upon panic, reboot, and recovery. The transport process need not depend on any "in memory" replication information other than the information in the save volume, so as

to permit the replication process to be started or terminated easily on any data mover for load shifting or load balancing.

When a save volume is used, it can be shared between a primary data mover and a secondary data mover in the case of local file system replication, or a primary copy of the shared volume can be kept at the primary site, and a secondary copy of the shared volume can be kept at the secondary site, in the case of remote file system replication.

For the case of local file system replication, FIG. 4 shows a primary site including a primary data mover 260 managing access to a primary file system 261, and a secondary data mover 262 managing access to a secondary file system 263 maintained as a read-only copy of the primary file system 261. A save volume 264 is shared between the primary data mover 260 and the secondary data mover 262. This sharing is practical when the secondary site is relatively close to the primary site. A redo log 265 records a log of modifications to the primary file system 261 during the replication process for additional protection from an interruption that would require a reboot and recovery.

Local replication can be used to replicate files within the same network file server. For example, in the network file server 221 in FIG. 1, the primary data mover could be the data mover 226, the secondary data mover could be the data mover 227, the save volume could be stored in the cached disk array 229, and replication control messages could be transmitted between the data movers over the data link 231.

For the case of remote file system replication, FIG. 5 shows a primary site including a primary data mover 270 managing access to a primary file system 271, and a secondary data mover 272 managing access to a secondary file system 273 maintained as a read-only copy of the primary file system 271. The primary site includes a primary

save volume 274, and the remote site includes a secondary save volume 275. A redo log 276 records a log of modifications to the primary file system 271 during the replication process for additional protection from an interruption that would require a reboot and recovery.

FIG. 6 shows a method of operating the system of FIG. 4 for local replication. In a first step 281, the primary data mover migrates a copy of the primary file system to create a secondary file system at the secondary site in such a way to permit concurrent write access to the primary file system. The migration, for example, may use the method shown in FIG. 17 of Ofek U.S. Patent 5,901,327, in which a bit map indicates remote write pending blocks. Alternatively, the migration may use a snapshot copy mechanism, for example, as described in Kedem, U.S. Patent 6,076,148, in which a bit map indicates the blocks that have changed since the time of snap-shotting of the primary file system. The snapshot method is preferred, because it is most compatible with the delta set technique for remote copy of subsequent modifications. For example, a snapshot manager creates a snapshot copy of the primary file system, as will be further described below with reference to FIGS. 25 to 27. The migration may involve taking a first snapshot copy of the primary file system, copying the first snapshot copy to the secondary file system, starting replication and creating a second snapshot copy of the primary file system, and then copying the differential between the first snapshot copy and the second snapshot copy from the primary file system to the secondary file system, in a fashion similar to that described further below with reference to steps 536 and 537 in FIG. 28. In any event, it is desired for the secondary file system to become a copy of the state of the primary file system existing at some point of time, with any subsequent

modifications of the primary file system being transferred through the shared save volume.

In step 282, the primary data mover writes subsequent modifications of the primary file system to the shared save volume. In step 283, the secondary data mover reads the subsequent modifications from the shared save volume and writes them to the secondary file system. In step 284, the secondary data mover provides user read-only access to consistent views of the secondary file system, by integrating the subsequent revisions into the secondary file system while providing concurrent read-only access to the secondary file system. This can be done by using a remote data facility to maintain a remote copy of a pair of delta volumes and to switch between the delta volumes as described in Suchitra Raman, et al., U.S. Patent Application Ser. No. 10/147,751 filed May 16, 2002, entitled "Replication of Remote Copy Data for Internet Protocol (IP) transmission, incorporated herein by reference. This can also be done by successively accumulating delta sets at the secondary, and accessing a block index or map of updated data blocks in the delta set.

Each delta set, for example, is a set of consistent updates to blocks of the secondary file system. The updated blocks are included in "delta chunks" transmitted from the primary file server to the secondary file server. Each delta set includes a number of delta chunks, and each delta chunk belongs to a single delta set. Each delta chunk is transmitted with a header including a generation count, a delta set number, and a chunk number within the delta set. When a delta set is accumulated, a block index or map is created to identify the updated blocks in the delta set. Once the delta set has been accumulated, a background integration process is started that writes the updates into the

secondary file system, and the progress of the background process is recorded by updating the block index or map for the delta set when each updated block is written to the secondary file system. During the background process, read access to the secondary file system is permitted on a priority basis. For this read access to a specified block of the file system, the secondary file server first accesses the block index or map of the delta set, and if the specified block is in the delta set, the secondary file server returns the data of the block from the delta set. If the specified block is not in the delta set, then the secondary file server returns the data of the block from the secondary file system.

In FIG. 6, execution loops from step 284 back to step 282. In this fashion, the secondary file system is updated from the primary site concurrently with read-only access at the secondary site.

FIG. 7 shows a method of operating the system of FIG. 5 for remote replication. In a first step 291, the primary data mover migrates a copy of the primary file system to create a secondary file system at the secondary site, in a fashion similar to step 281 in FIG. 6. In step 292, the primary data mover writes subsequent modifications of the primary file system to the primary save volume, in a fashion similar to step 282 in FIG. 6. In step 293, the modifications are copied from the primary save volume to the secondary save volume, by transmitting delta chunks. In step 294, the secondary data mover reads the modifications from the secondary save volume and writes them to the secondary file system. In step 295, the secondary data mover provides user read-only access to consistent views of the secondary file system, in a fashion similar to step 284 of FIG. 6. Execution loops from step 295 back to step 292. In this fashion, the secondary file

system is remotely updated from the primary site concurrently with read-only access at the secondary site.

FIG. 8 shows layered programming 300 for a primary data mover. It is desired to use layered programming in accordance with the International Standard Organization's Open Systems Interconnection (ISO/OSI) model for networking protocols and distributed applications. As is well known in the art, this OSI model defines seven network layers, namely, the physical layer, the data link layer, the network layer, the transport layer, the session layer, the presentation layer, and the application layer.

As shown in FIG. 8, the layered programming 300 includes a conventional TCP/IP transport layer 301. The layers above the TCP/IP transport layer 301 include a replication control protocol (RCP) session layer 302, a volume multicast presentation layer 303, and an IP-FS (file system) copy send-thread 304 and an IP-replication send-thread 305 at the program layer level. Over these program layers is a management and configuration command interpreter (MAC_CMD) 306 for system operator set-up, initiation, and supervisory control of the replication process.

In operation, the RCP layer 302 provides an application program interface (API) for multicasting data over TCP/IP. RCP provides callback, acknowledgement (ACK), and resumption of aborted transfers.

RCP provides the capability for a remote site to replicate and rebroadcast remote copy data. The remote site functions as a router when it rebroadcasts the remote copy data. RCP can also be used to replicate data locally within a group of data movers that share a data storage system.

To create a new remote copy in response to a supervisory command, the command interpreter 306 initiates execution of a replication module 310 if the replication module is not presently in an active mode. Then, the command interpreter 306 invokes a snapshot manager 308 to create a snapshot copy 309 of a primary file system volume 307. When the snapshot copy is created, the snapshot manager 308 obtains a current delta set number from the replication module 310 and inserts the current delta set number into the metadata of the snapshot. The current delta set number for the snapshot is all that the secondary needs to identify modifications that are made subsequent to the creation of the snapshot. In this fashion, any number of new remote copies can be created at various times during operation of the replication module, with the snapshot process operating concurrently and virtually independent of the replication module. For example, whenever synchronization of a remote copy is lost, for example due to a prolonged disruption of network traffic from the primary site to the remote site, a new remote copy can be created to replace the unsynchronized remote copy.

Once the snapshot copy 309 is accessible, the command interpreter 306 initiates execution of an instance of the IP-FS copy send-thread 304. The instance of the IP-FS copy send-thread 304 reads data from the snapshot copy 309 and calls upon the volume multicast layer 303 to multicast the remote copy data to all of the secondary data movers where the remote copies are to be created. This can be a copy by extent, so there is no copying of invalid or unused data blocks. For example, the volume multicast layer 303 is given a copy command (@vol., length) specifying a volume and an extent to be copied, and may also specify a group of destinations (an RCP group). The snapshot copy 309 of the primary file system identifies the next valid block to be copied, and the number of

valid contiguous blocks following the next block. These blocks are copied at the logical level, so it does not matter what physical structure is used for storing the secondary file system at the secondary site. The copying is done locally, or by remote copy, for example by transporting the data block over IP. The volume multicast layer 303 invokes the RCP layer 302 to transport each data block.

During the remote copy process, whenever a modification is made to a block of the primary file system volume 307, the replication module 310 logs an indication of the modified block in a log 314 and later assembles the modification into a delta set chunk written to a primary save volume 311. The replication module 310 logs the indications in the log 314 on a priority or foreground basis as data is written to the primary file system volume 307, and also logs boundaries between delta sets. The replication module 310 later reads the log 314 to read the indicated modifications from the primary file system volume 307, assemble the indicated modifications into delta set chunks on a background basis, and store the delta set chunks in a save volume chunk area of the save volume 311. For example, the log is in the form of a queue of two bit-map tables, a new one of the tables being written to coincident with write operations upon the primary file system volume 307, and an old one of the tables being read to determine blocks to copy from the primary file system to create a new delta set in the save volume 311. When the delta set chunks become available for distribution from the save volume 311, the replication module 310 updates the save volume mailbox area 312 by storing each delta set chunk definition (@vol., length).

The IP-replication send-thread instance 305 polls the save volume mailbox area 312 to see if any delta set chunks have been stored in the save volume chunk area 313. If

so, then the thread instance calls upon the volume multicast layer 303 to multicast the delta set chunks to the data movers that manage the storage of the respective remote file system copies. For example, for each delta set chunk, the IP-replication send-thread instance 305 issues a volume multicast command to the volume multicast layer 303. When the chunk multicast is completed, the IP-replication send-thread instance 305 updates its context on the save volume 311 in the mailbox area 312. At reboot after an interruption of multicast of a chunk, the IP-replication send-thread instance is able to restart the multicast of the chunk. The IP-replication send-thread instance also is responsible for retrying transmission of the chunk whenever the connection with the secondary is interrupted.

FIG. 9 shows the layered programming 320 for a secondary data mover. The programming includes a TCP/IP layer 321, an RCP layer 322, a volume multicast layer 323, and a management and configuration command interpreter (MAC_CMD) 324. During creation of a new remote copy in a secondary file system volume 325, the volume multicast layer 323 writes remote copy data from the primary data mover to the secondary file system volume 325, and concurrently writes modifications (delta set chunks) from the primary data mover to a save volume chunk area 326 of a secondary save volume 327.

A header for the changes in a next version of the delta set is sent last, because there is no guarantee of the order of receipt of the IP packets. The header of the delta set includes a generation count, the number of delta blocks for the next version of the delta set, a checksum for the header, and a checksum for the data of all the delta blocks. The receiver checks whether all of the changes indicated in the header have been received.

Once a complete remote snapshot copy has been reconstructed in the secondary file system volume 325, a playback module 328 is activated to read the modifications from the save volume chunk area 326 and integrates them into the secondary file system volume 325. From each delta-set chunk in the save volume area 326, the playback module 328 gets the block address and number of contiguous blocks to be written to the secondary file system volume. An access module 329 provides read-only access to a consistent view of the secondary file system in the secondary file system volume 325.

FIG. 10 shows a procedure executed by the primary site of FIG. 8 to perform replication of the primary file system. When replication is started in a first step 341, the primary file system is paused to make it consistent. Migration of the primary file system to the secondaries can then be started using a remote copy facility or snapshot manager. Then, in step 342, concurrent write access to the primary file system is resumed, and all modifications made on the primary file system are logged at the volume level on a priority or foreground basis when each modification is made. In addition, a background process of delta-set creation is initiated.

Two configurable triggers specify the rate of delta set creation: a timeout parameter and a high water mark parameter. Whenever delta set creation is initiated, the current time, as indicated by a real-time clock, is added to a configurable timeout interval to produce the timeout parameter. The high water mark specifies an amount of modified data, in megabytes. The first trigger that occurs will trigger the creation of a delta set. The replication module creates the delta set by pausing the primary file system, copying the modified blocks from the primary file system to the delta set volume, and then resuming the primary file system. By logging indications of the modified blocks and

later copying the modified blocks, multiple modifications to the same block are represented and transported once during a single delta set.

In step 343, the background process of delta set creation is temporarily suspended, for example, by placing the process on a task queue that is periodically serviced. In step 344, execution of the delta set creation process is resumed. In step 345, the modification size is compared to the high water mark. If the high water mark is not exceeded, then execution continues to step 346. In step 346, the present value of the real-time clock is compared to the timeout parameter. If the timeout parameter has not been exceeded, then execution loops back to step 343. Otherwise, execution continues to step 347. Execution also branches to step 347 from step 345 if the modification size is greater than the high water mark.

In step 347, the primary file system is paused. In step 348, a new delta set is created by starting the copying of modified blocks from the primary file system volume to the new delta set. In step 349, the logging of new modifications into a new table is started. In step 350, the time-out and high water mark is re-armed. In other words, a new value for the timeout parameter is computed as the current real time plus the configurable timeout interval, and the modification size is reset to indicate the size of the new modifications. In step 351, the primary file system is resumed. Execution loops from step 351 back to step 343 to suspend the background process of delta set creation.

To maintain the consistency of the delta set created in the primary save volume, the primary file system could remain paused and not resumed in step 351 until the copy process begun in step 348 is completed. Preferably, however, the copy process begun in step 348 is a snapshot copy process, so that write access to the primary file system may

resume in step 351 before the copy process has been completed. For the example of the modification log being a queue of two bit-map tables, when a write access to a block in the primary file system is requested, the old bit map is accessed on a priority basis. If the corresponding bit in the old bit map indicates a modified block in the primary file system volume not yet copied to the save volume, then it is copied on a priority basis to the save volume before the new write data is written to the primary file system volume. As soon as a modified block has been copied from the primary file system volume to the save volume, the corresponding bit in the old bit map is cleared. In this fashion, at the completion of the copy process, the entire old table will be in a reset state, ready to be used as the next new table.

When the copy process started in step 348 is completed, the replication module sets the save volume mailbox area to show that a new delta set is ready for transmission. Upon polling the mailbox area, the IP-replication send-thread finds that the new delta set is ready for transmission, and invokes the volume multicast layer to transmit the delta set to the secondary sites. After step 351, execution loops back to step 343.

FIG. 11 shows a flow chart of the overall procedure of creating a new remote copy, either for the first time at a secondary site or as a replacement for a remote copy that needs to be resynchronized with the primary file system. In a first step 352, the snapshot manager creates a snapshot copy of the primary file system at the end of any pending transaction upon the primary file system (e.g., when the primary file system becomes consistent after it is paused in step 341 of FIG. 10 or in step 347 of FIG. 10.) The replication module independently writes any subsequent modifications into a current delta set for the next transaction.

In step 353, the snapshot manager obtains the current delta set number from the replication module and inserts it into metadata of the snapshot copy. In step 354, the IP-FS copy send-thread is started in order to send volume extents of the snapshot copy to the secondary data mover, by invoking the volume multicast layer for each extent.

In step 355, when the IP-FS copy send-thread is finished, the primary data mover sends a "start playback" signal to the secondary data mover. In step 356, the secondary data mover receives the "start playback" signal from the primary data mover, and starts the playback module. In step 357, playback module begins playback from the delta set indicated by the delta set number in the snapshot metadata.

The playback module (328 in FIG. 23) at the secondary site integrates the delta set modifications into secondary file system. Each time that a new delta set appears in the secondary save volume, the modifications can be integrated into the secondary file system, for example, by pausing the secondary file system, copying the modifications from the secondary save volume into the secondary file system, and resuming the secondary file system. Alternatively, a timeout interval and a high water mark value can be configured for the secondary site, so that the modifications may be integrated into the secondary file system at a rate less frequent than the rate at which the new delta sets appear in the secondary save volume. In this case, the modifications from the secondary save volume would not be integrated into the secondary file system until the timeout time is reached unless the amount of modifications in the save volume reaches the high water mark. As described above, the integration of the modifications can be performed concurrently with read-only access to a consistent view of the secondary file system.

FIG. 12 shows a flowchart of the IP-replication send-thread (305 in FIG. 8). In a first step 361, the thread polls the primary save volume mailbox area. If the mailbox area indicates that there is not a new delta set chunk in the primary save volume area, then the thread is finished for the present task invocation interval. Execution of the thread is suspended in step 363, and resumed in step 364 at the next task invocation interval.

If the mailbox area indicates that there is a new delta set chunk in the primary save volume, then execution continues from step 362 to step 365. In step 365, the IP-replication send-thread issues a volume multicast command to broadcast or forward the delta set chunk to specified destination data movers. In step 366, if the multicast has been successful, then execution branches to step 367. In step 367, the IP-replication send-thread updates the primary save volume mailbox to indicate completion of the multicast, and execution continues to step 363 to suspend execution of the thread until the next task invocation interval.

In step 366, if the multicast is not successful, then execution continues to step 368 to test whether more than a certain number (N) of retries have been attempted. If not, then execution loops back to step 365 to retry the multicast of step 365. If more than N retries have been attempted, then execution continues from step 368 to step 369. In step 369, the IP-replication send-thread logs the error, and then in step 370, passes execution to an error handler.

FIG. 13 shows various objects defined by the volume multicast layer. The volume multicast layer provides multicast service to instances of a VolMCast object 370 representing a volume multicast relationship between a respective primary file system volume specified by a volume name (volumeName) and a respective group of secondary

data movers specified by an RCP group name (rcpgpeName). For example, at configuration time, one or more RCP groups are defined in response to configuration commands such as:

.RCP_config <server_name> add <IP>

This configuration command adds the IP address (IP) of a specified destination data mover (server_name) to an RCP group.

Also at configuration time, a specified data mover can be defined to be a primary data mover with respect to the RCP group (a relationship called a MultiCastNode) in response to a configuration command such as:

.server_config <server_name> rep <groupname> add <IP>

where "server_name" is the name for the primary data mover, "groupname" is the name of a configured RCP group, and "IP" is the IP address of the primary data mover. When configuration of the MultiCastNode object is finished, the MultiCastNode object will have its own name, a name for the primary data mover, an RCP group name, and a list of IP addresses to which the primary server should broadcast in order to transmit IP packets to all the secondary data movers in the RCP group.

The VolMCast object can then be built on top of a MultiCastNode object. The additional information required for the VolMCast object is, on the sender side, the primary or source file system volume and on each receiver side, the secondary or

destination file system volume. For flexibility, it is permitted to specify a different volume name on each secondary data mover. By specifying the destination volume names during creation of the VolMCast object, it is not necessary to specify the destination volume names at each copy time. For example, the VolMCast object is defined by configuration commands to the primary data mover such as:

.server_config <server_name> "volmcast MultiCastNodeName>
[-src | -dest] volume"

where <server_name> is the name of the MultiCast Node.

Once the VolMCast object has been defined, an IP-replication service can be configured for the object upon the primary data mover. Then the primary data mover will respond to commands for starting the replication service and stopping the replication service upon the VolMCast object. When replication is stopped on a secondary, the secondary file system is left in a consistent state. In other words, if a replay was in progress, the stop will complete when the replay is finished.

The primary data mover may respond to additional commands for create a new delta set on demand, updating the replication policy (high water mark and timeout interval parameters) on the primary file system or secondary file systems, and defining persistency of the replication process upon remount or reboot of the primary file system or any one of the secondary file systems. For example, at reboot the replication service is re-started on the primary file system and the secondary file system in the state it was at unmount or shutdown. A recovery of the replication context happens at reboot or on

remount. The replica recovery is executed before the primary and secondary file systems are made available for user access. This allows all modifications during the recovery of the primary file system to be logged by the replication service.

As shown in FIG. 13, the volume multicast layer is responsive to a number of commands 371 from higher layers in the protocol stack. In addition to the configuration commands for defining a new VolMCast object relating a specified primary file system volume to a specified RCP group, an existing VolMCast object can be opened for either a sender mode or a receiver mode. An opened VolMCast object can be closed. Once a VolMCast object has been opened in a sender mode, it can be called upon to broadcast a control block (CB) to the secondary volumes of the VolMCast object, such as a control block specifying a remote copy of a specified extent of the primary volume.

Control blocks may specify various operations upon the secondary volumes of the VolMCast object, such as cluster file system commands for performing operations such as invalidations, deletions, renaming, or other changes in the configuration of the objects of the file system upon all copies (local or remote) of the file system. In this case, RCP is used for the broadcast or forwarding of the cluster file system commands to all the data movers that are to operate upon the local or remote copies of the file system, and for returning acknowledgement of completion of the operations upon all of the copies of the file system.

With reference to FIG. 13, the volume multicast layer defines a VolMCastSender object 372 instantiated when a VolMCast instance is opened in the sending mode, and a VolMCastReceiver object 373 instantiated when a VolMCast instance is opened in a receiving mode. The VolMCastSender object class and the VolMCastReceiver object

class inherit properties of the VolMCast object class. When the volume multicast layer is called upon in a primary data mover to maintain remote copies of a specified extent of a VolMCastSender instance, an instance of a VolMCastCopy thread 374 is created and executed. The VolMCastCopy thread instance accesses the delta sets from a primary save volume 375 to produce a write stream 376 of blocks sent down to the RCP layer. At the secondary data mover, an instance of a VolMCastReceiver thread 377 is instantiated and executed to receive a read stream 378 of blocks and write the copied delta sets into a secondary save volume 379. An instance of an acknowledgement thread 380 returns an acknowledgement 381 of completion of copying of a delta-set for an extent to the secondary file system. The acknowledgement is sent down to the RCP layer of the secondary data mover. At the primary, the RCP layer sends the acknowledgement 382 to an instance of an acknowledgement thread 383.

RCP is a session-layer protocol, for replication from one primary to multiple secondary sites. Control is initiated by the primary, except when recovering from aborted transfers. RCP uses TCP between the primary and secondary for control and data. Network distribution is by an application-level multicast (ALM) using the RCP as a forwarder. Port sharing with HTTP is used for crossing firewalls.

RCP may support other replication applications in addition to 1-to-N IP-based replication for wide-area distribution of read-only data. These other applications include 1-to-N volume mirroring, cluster file system commands, remote file system replication, and distribution and replication of other commands that may be recognized by the data movers.

The 1-to-N volume mirroring is a simplification of to 1-to-N IP-based replication for wide-area distribution of read-only data, because the volume mirroring need not synchronize a remote volume with any consistent version of the primary volume until the remote volume needs to be accessed for recovery purposes.

Remote file system replication also uses RCP for broadcast or forwarding an application command to a remote data mover to initiate a replication of a file system managed by the remote data mover. In a similar fashion, RCP may broadcast or forward other commands recognized by data movers, such as iSCSI or remote-control type commands for archival storage. For example, RCP could broadcast or forward remote control commands of the kind described in Dunham, U.S. Patent 6,353,878 issued March 5, 2002 entitled "Remote Control of Backup Media in a Secondary Storage Subsystem Through Access to a Primary Storage Subsystem," incorporated herein by reference.

The RCP forwarder is composed of two RCP sessions, an outbound session at the primary, and an inbound session at the secondary. The inbound RCP session receives a group name and looks up the group in a routing table. If routes for the group exist in the routing table, then an RCP forwarder is created at the secondary, including a data path by pointer passing from an "in" session to an "out" session.

An RCP group may be configured to include application-level multicast (ALM) topology. For example, ALM route configuration commands begin with an identifier number for the network file server ("cel") that contains the forwarder data mover, and an identifier number ("ser") for the forwarder data mover in the network server. The configuration commands end with a "nexthop" specification of an immediate destination data mover:

cell-ser2: rcproute add group=g1 nexthop=cell2-ser2

cel2-ser2: rcproute add group=g1 nexthop=cell2-ser3

cel2-ser2: rcproute add group=g1 nexthop=cell2-ser4

In effect, the forwarder data mover adds the "nexthop" specification to an entry for the RCP group in the routing table in the forwarder data mover. This entire entry can be displayed by the following configuration command:

cel2-ser2: rcproute display

The entry is displayed, for example, as a list of the "nexthop" destination data movers. The entry can be deleted by the following configuration command:

cel2-ser2: rcproute delete

Each immediate destination data mover may itself be configured as a forwarder in the RCP group. In this case, RCP commands and data will be forwarded more than once, through a chain of forwarders. The set of possible RCP routes from a primary or forwarder in effect becomes a tree or hierarchy of destinations.

The ALM commands may also include commands for creating sessions and sending control blocks or data. For example, the following ALM command creates a session and sends application data to all destinations in group "g1" from cell1-ser2 from a file (named "filename") using a test application (named "rcpfiletest").

cell-ser2: rcpfiletest data=filename group=g1

FIG. 14 shows the RCP collector service 390 at a primary site. The programming for the RCP collector service includes an RCP session manager 391, collector and worker threads 392, and a single-thread RCP daemon 393. The RCP session manager 391 responds to requests from higher levels in the protocol stack, such as a request from an application 394 to open an RCP pipe 395 between the application 394 and the RCP collector service 390. The application 394 may then send to the session manager 391 requests to setup sessions with RCP groups. A session queue 396 stores the state of each session, and a control block queue 397 keeps track of control blocks sent via TCP/IP to the secondary data movers in the RCP groups. An RCP routing table 398 identifies the immediate destinations of each RCP group to which the TCP/IP messages from the RCP collection service are to be sent, as well as any other destinations to which the messages will be forwarded. For communication of the TCP/IP messages between the RCP service and the network, TCP port :80 is opened in both directions (i.e., for input and output). The single thread RCP daemon 393 is used for interfacing with this TCP port :80.

FIG. 15 shows the RCP collector service 400 at a secondary site. The RCP collector service at the secondary site is similar to the RCP collector service at the primary site, in that it includes an RCP session manager 401, collector and worker threads 402, a single thread RCP daemon 403 for access to/from TCP port :80, an RCP session state queue 406, an RCP control block queue 407, and an RCP routing table 408. The primary difference between the RCP collector service at the secondary site from the

RCP collector service at the primary site is in the collector and worker threads 402. At the RCP secondary, the RCP commands and data are received from the TCP port :80 instead of from the application 404. The application 404 is the consumer of the RCP data, instead of a source for RCP data. The RCP collector service 400 at the secondary site may also serve as a forwarder for RCP commands, and therefore the RCP collector service and worker threads 402 at the secondary site include a forwarder thread that does not have a similar or complementary thread in the RCP collector service at the primary site.

In operation, an application 404 can initialize the RCP collector service so that the RCP collector service will call back the application upon receipt of certain RCP commands from TCP port :80. For example, if a new connection command is received from TCP port :80, then the RCP daemon 403 forwards the new connection command to the RCP session manager. The RCP session manager 401 recognizes that this connection command is associated with an application 404 at the secondary site, opens an RCP pipe 405 to this application, and calls the application 404 indicating that the RCP pipe 405 has been opened for the RCP session. (The volume multicast receiver thread 377 of FIG. 13 is an example of such an application.) The application 404 returns an acknowledgement. If the new connection is for a new RCP session, then the session manager creates a new RCP session, and places state information for the new session on the RCP session queue 406. RCP control blocks and data may be received for the session from the TCP port :80. The data may be forwarded to the application, or to a file specified by the application. RCP control blocks to be executed by the RCP collector service 400 may be temporarily

placed on the control block queue 407. RCP control blocks or data intended for other secondary site may be forwarded to the intended secondary sites.

FIG. 16 shows further details of the forwarding of RCP commands and data by a data mover 430 identified as Cel2-Ser1. The data mover 430 is programmed with a TCP/IP layer 431 for communication with the IP network 220, and an RCP layer 432 over the TCP/IP layer. For forwarding the RCP commands and data, the RCP layer 432 creates an inbound session 433 and an outbound session 434. The inbound session 433 receives RCP commands from the TCP/IP layer 431. The TCP/IP data stream is retained in a data buffer 435. When an RCP command calls for the forwarding of RCP commands or data to another data mover in a specified RCP group, the inbound session 433 performs a lookup for the group in a routing table 436.

In the example of FIG. 16, the routing table 436 includes a copy of all of the routing information for each group of which the data mover 430 is a member. In this case, for GROUP1, the primary data mover sends RCP commands to at least data movers CEL2-SER1 and CEL9-SER1. CEL2-SER1 (i.e., the data mover 430) forwards the RCP commands and RCP data to data movers CEL3-SER1 and CEL7-SER1. In particular, the inbound session 433 creates an outbound session 434 and creates a TCP/IP data path from the inbound session 433 to the outbound session 434 by passing pointers to the data in the data buffer. The outbound session 434 invokes the RCP/IP layer 431 to multicast the TCP data stream in the data buffer 435 over the IP network 220 to the data movers CEL3-SER1 and CEL7-SER1.

The data mover CEL3-SER1 in succession forwards the RCP commands to data movers CEL4-SER1 and CEL5-SER1. Normally, the data mover CEL2-SER1 (430)

does not need to know that the data mover CEL3-SER1 forwards the RCP commands to data movers CEL4-SER1 and CEL5-SER1, but if the data mover CEL2-SER1 (430) would fail to receive an acknowledgement from CEL3-SER1, then the data mover CEL2-SER1 could minimize the impact of a failure of CEL3-SER1 by forwarding the RCP commands to CEL4-SER1 and CEL5-SER1 until the failure of CEL3-SER1 could be corrected.

FIG. 17 shows a flowchart of how the RCP collector service at the secondary site processes an inbound RCP session command. In a first step 411, the RCP collector service receives a session command. In step 412, if this session command is not a command to be forwarded to other secondary sites, then execution branches to step 413 to execute the action of the command, and the processing of the session command is finished.

In step 412, if the session command is a command to be forwarded to other secondary sites, then execution continues from step 412 to step 414. In step 414, the RCP collector service gets the RCP group name from the session command. Then, in step 415, the RCP collector service looks up the group name in the RCP routing table (408 in FIG. 15). If the group name is not found, then execution branches from step 416 to step 417. In step 417, the RCP collector service returns an error message to the sender of the session command.

In step 416, if the group name is found in the RCP routing table, then execution continues from step 416 to step 418. In step 418, the RCP collector service forwards the action of the session command to each secondary in the group that is an immediate destination of the forwarder (i.e., the data mover that is the secondary presently

processing the RCP session command). This is done by instantiating local replication threads or creating outbound sessions for forwarding the action of the session command to each secondary in the group that is an immediate destination of the forwarder. After step 418, processing of the RCP session command is finished.

FIG. 18 shows an example of forwarding and local replication. In this example, the IP network 220 connects a primary data mover 421 to a network file server 422 and a secondary data mover 423. The network file server 422 includes three data movers 424, 425, and 426, and storage 427. The primary data mover manages network access to a primary file system 428. The data mover 424 functions as a forwarder data mover. The data mover 425 functions as a secondary data mover managing access from the network to a secondary file system (copy A) 429. The data mover 426 functions as a secondary data mover managing access from the network to a secondary file system (copy B) 430. The data mover 423 manages network access to a secondary file system (copy C) 431.

In operation, when the primary data mover 421 updates the primary file system 428, it multicasts the modified logical blocks of the file system volume over the IP network 220 to the forwarder data mover 424 and to the secondary data mover 423. The forwarder data mover 424 receives the modified blocks, and performs a local replication of the blocks to cause the secondary data mover 425 to update the secondary file system (copy A) 429 and the to cause the secondary data mover 426 to update the secondary file system (copy B) 430.

To perform the local replication, the forwarder data mover 424 has its volume multicast layer (323 in FIG. 9) save the modified blocks in a save volume 432 in the storage 427, and then the forwarder data mover 424 sends replication commands to the

local secondary data movers 425 and 426. Each local secondary data mover 425, 426 has its playback module (328 in FIG. 9) replay the modifications from the save volume 432 into its respective secondary file system copy 429, 430.

FIG. 19 shows the sharing of the data mover's network TCP port :80 (440) between HTTP and RCP. This configuration is used in all data movers having the RCP collector service; i.e., primary, secondary, or forwarder. The TCP data channel from TCP port :80 (440) provides an in-order byte stream interface. IP packets 444 for HTTP connections and IP packets 445 for RCP connections from the network 220 are directed to the data mover's TCP port :80 (440). The TCP port :80 (440) is opened in both directions (i.e., input and output). In the input direction, the data mover uses a level 5 (L5) filter 441 for demultiplexing the IP packets for the HTTP connections from the IP packets for the RCP connections based on an initial segment of each TCP connection. The L5 filter hands the TCP connection off to either a HTTP collector service 442 or an RCP collector service 443. (The RCP collector service 443 is the collector service 390 in the RCP primary of FIG. 14 or the RCP collector service 400 in an RCP secondary of FIG. 15.) For example, if the initial segment of a TCP connection contains "HTTP/1.X", then the L5 filter 441 directs the IP packets for the connection to the HTTP collector service 442. If the initial segment of the TCP connection contains "RCP/1.0", then the IP packets for the TCP connection are directed to the RCP collector service 443. (In an alternative arrangement, the connection could be split as is done in a conventional stand-alone IP switch.)

Data Recovery With Internet Protocol Replication With Or Without Full Resync

A remote replication system may protect against data loss resulting from a disaster. For example, FIG. 20 shows the state of a remote replication system before a disaster. The remote replication system copies data in an asynchronous fashion from a primary data storage system at the primary site to a secondary data storage system at a secondary site. The primary and secondary data storage systems could be file servers, for example as shown in FIG. 1.

As shown in FIG. 20, the primary site includes a replication service 450 replicating changes made to a primary file system 451, and the secondary site includes a playback service 452 writing the changes into a secondary file system 453. The replication service 450 transfers the changes in "delta sets" sent over an IP pipe 454 to the playback service 452. The disaster occurs at the primary site during this transfer. At this time, the playback service 452 is a few delta sets behind the replication service 450; for example, the playback service is playing back delta set (n-x) when the replication service 452 is replicating delta set (n).

Once the disaster causes the primary site to go down, the secondary site can be activated to service client read/write requests. When the primary site comes back up, it can be re-synchronized to the secondary site, in many cases without making a full copy of the file system. Then replication can be re-established as it was before the disaster. This recovery process has been designed to minimize data loss and the down time of both the primary and secondary site.

FIG. 21 is a flowchart of the preferred recovery process. Initially, in step 461, data is replicated from the primary file system at the primary site by sending delta sets to the secondary file system at the secondary site. Then in step 462, the primary site becomes inoperative. In response, in step 463, the secondary site is activated for read/write access to the secondary file system. This is done by a subroutine for failover with a checkpoint and without sync, as further described below with respect to FIG. 23. Eventually, in step 464, the primary site becomes operative. In response, in step 465, the primary file system is synchronized to the state of the secondary file system, as further described below with respect to FIG. 28. Then in step 466, read/write access to the primary file system and replication of data from the primary file system to the secondary file system is resumed in a failback operation, as further described below with reference to FIG. 30.

FIG. 22 shows the state of the remote replication system of FIG. 20 during the failover operation. The (n-x) deltasets are played back into the secondary file system 453, to put the secondary file system 453 into a state called a "restart point". A snapshot copy facility 456 creates a snapshot 455 of the "restart point" in order to facilitate synchronization of the primary file system with the secondary file system once the primary site becomes operative. The playback service 452 is stopped, and the secondary file system 453 is remounted as read/write. The snapshot copy facility 456 keeps a record of changes made to the secondary file system 453 after the "restart point."

FIG. 23 shows a flowchart of the failover operation. In step 471, if possible, the primary file system is remounted as a read-only file system. Next, in step 472, the secondary site finishes playback of the (n-x) delta sets that it had received but had not

played back prior to the primary site becoming inoperative. In step 473, the secondary site creates a snapshot copy of the "restart point" of the secondary file system. This snapshot is stamped with the last delta set having been replayed into the secondary file system prior to the creation of the snapshot. In other words, an identifier of this last delta set is stored as an attribute of the snapshot. In general, whenever a snapshot is made of a file system during playback of delta sets into the file system, the snapshot is stamped with the last delta set having been replayed. In step 474, the secondary file system is remounted as a read/write file system. In step 475, the snapshot process retains and identifies all changes (delta) made to the secondary file system since the restarting point.

Preferably the snapshot copy facility 456 retains and identifies the changes at a logical volume level of data storage. For example, the present state of the secondary file system is stored in a "clone volume," and old versions of the logical blocks that have been changed in the clone volume are saved in a "save volume". In order to conserve storage, the logical blocks of the save volume are dynamically allocated to the old versions of the changed blocks as the changes are made to the clone volume. As shown in FIG. 24, for each logical block that has been changed in the clone volume, a block map 480 identifies the logical block address (S_i) of the old version of the block in the save volume and the corresponding logical block address (B_i) of the changed block in the clone volume.

FIG. 25 shows details of the preferred snapshot copy facility 456, which provides multiple snapshots 483, 503 of a production file system 481. (Two successive snapshots of the secondary file system are used, for example, in the resync subroutine further described below with reference to FIG. 28.) The content of each snapshot file system

483, 503 is the state of the production file system 481 at a particular point in time when the snapshot was created. The snapshot copy facility 456 provides a hierarchy of objects in a volume layer 490 supporting the file systems in a file system layer 491. The production file system 481 is supported by read/write access to a file system volume 482. Each snapshot file system 483, 503 provides read-only access to a respective snapshot volume 484, 504.

Additional objects in the volume layer 490 of FIG. 25 permit the content of each snapshot file system to be maintained during concurrent read/write access to the production file system 481. The file system volume 482 is supported by a snapped volume 485 having read access to a clone volume 487 and write access to a delta volume 486. The delta volume 486 has read/write access to the clone volume 487 and read/write access to a save volume 488.

In the organization of FIG. 25, the actual data is stored in blocks in the clone volume 487 and a respective save volume 488, 506 in storage for each snapshot. The delta volume 486 also accesses information stored in a bit map 489 and the block map 480. The bit map 489 indicates which blocks in the clone volume 487 have prior versions in the save volume 488. In other words, for read-only access to the snapshot file system, the bit map 489 indicates whether the delta volume should read each block from the clone volume 487 or from the save volume 488. For example, the bit map is stored in memory and it includes a bit for each block in the clone volume 487. The bit is clear to indicate that there is no prior version of the block in the save volume 488, and the bit is set to indicate that there is a prior version of the block in the save volume 488.

Consider, for example, a production file system 481 having blocks *a*, *b*, *c*, *d*, *e*, *f*, *g*, and *h*. Suppose that when the snapshot file system 483 is created, the blocks have values *a0*, *b0*, *c0*, *d0*, *e0*, *f0*, *g0*, and *h0*. Thereafter, read/write access to the production file system 481 modifies the contents of blocks *a* and *b*, by writing new values *a1* and *a2* into them. At this point, the following contents are seen in the clone volume 487 and in the save volume 488:

Clone Volume: *a1*, *b1*, *c0*, *d0*, *e0*, *f0*, *g0*, *h0*

Save Volume: *a0*, *b0*

From the contents of the clone volume 487 and the save volume 488, it is possible to construct the contents of the snapshot file system 483. When reading a block from the snapshot file system 483, the block is read from the save volume 488 if found there, else it is read from the clone volume 487.

FIG. 25 further shows that a snapshot queue 500 maintains respective objects supporting multiple snapshot file systems 483, 503 created at different respective points in time from the production file system 481. In particular, the snapshot queue 500 includes a queue entry (J+K) at the tail 501 of the queue, and a queue entry (J) at the head 502 of the queue. In this example, the snapshot file system 483, the snapshot volume 484, the delta volume 486, the save volume 488, the bit map 489, and the block map 480 are all located in the queue entry at the tail 501 of the queue. The queue entry at the head of the queue 502 includes similar objects; namely, the snapshot file system (J) 503, a

snapshot volume 504, a delta volume 505, a save volume 506, a bit map 507, and a block map 508.

The snapshot copy facility 456 may respond to a request for another snapshot of the production file system 481 by allocating the objects for a new queue entry, and inserting the new queue entry at the tail of the queue, and linking it to the snapped volume 485 and the clone volume 487. In this fashion, the save volumes 488, 506 in the snapshot queue 500 are maintained in a chronological order of the respective points in time when the snapshot file systems were created. The save volume 506 supporting the oldest snapshot file system 503 resides at the head 502 of the queue, and the save volume 488 supporting the youngest snapshot file system 483 resides at the tail 501 of the queue.

FIG. 26 shows a procedure performed by the snapshot copy facility for writing a specified block (B_i) to the production file system. In step 511, if the snapshot queue is not empty, execution continues to step 512. In step 512, the bit map at the tail of the snapshot queue is accessed in order to test the bit for the specified block (B_i). Then in step 513, if the bit is not set, execution branches to step 514. In step 514, the content of the specified block (B_i) is copied from the clone volume to the next free block in the save volume at the tail of the snapshot queue. Execution continues from step 514 to step 515. In step 515, the save volume block address (S_i) of the free block is inserted into the entry for the block (B_i) in the block map at the tail of the queue, and then the bit for the block (B_i) is set in the bit map at the tail of the queue. After step 515, execution continues to step 516. Execution also continues to step 516 from step 513 if the tested bit is found to be set. Moreover, execution continues to step 516 from step 511 if the snapshot queue is

empty. In step 516, new data is written to the specified block (B_i) in the clone volume, and then execution returns.

FIG. 27 shows a procedure performed by the snapshot copy facility for reading a specified block (B_i) from a specified snapshot file system (N). In the first step 521, the bit map is accessed for the queue entry (N) to test the bit for the specified block (B_i). Then in step 522, if the tested bit is set, execution continues to step 523. In step 523, the block map is accessed to get the save volume block address (S_i) for the specified block (B_i). Then in step 524 the data is read from the block address (S_i) in the save volume, and then execution returns.

If in step 522 the tested bit is not set, then execution branches to step 525. In step 525, if the specified snapshot (N) is not at the tail of the snapshot queue, then execution continues to step 526 to perform a recursive subroutine call upon the subroutine in FIG. 27 for read-only access to the snapshot (N+1). After step 526, execution returns.

If in step 525 the snapshot (N) is at the tail of the snapshot queue, then execution branches to step 527. In step 527, the data is read from the specified block (B_i) in the clone volume, and execution returns.

FIG. 28 shows the subroutine to resync the primary file system with the secondary file system, as used in the flowchart of FIG. 21. In a first step 531 in FIG. 28, the delta set identifier (n-x) for the restart point is read from the delta set attribute of the restart point snapshot, and it is incremented by one to compute (n-x+1). The save volume at the primary site is searched for the delta set (n-x+1). Execution continues from step 531 to step 532 if the delta set n-x+1 is found in the save volume at the primary site. In this

case, the primary site should also have all of the delta sets from delta set n-x+1 to delta set n, and the primary file system can be restored to the state of the restart point in step 532 by an "undo" of the data blocks of these delta sets. The primary file system is restored with "before images" of these data blocks in order to "undo" the changes. The delta sets n-x+1 to n contain "after images" of these data blocks. The "before images" of these data blocks are stored in the secondary site in the snapshot of the secondary file system at the restart point. Therefore, to restore the primary file system to the state of the restart point, the snapshot copy facility at the secondary site reads the "before images" from the "restart point" snapshot, and the secondary site executes a remote copy of these "before images" to the primary file system.

In step 532, the primary site restores the primary file system to the state of the restarting point by obtaining a list of blocks from the save volume at the primary site, including the blocks in delta set n-x+1 to delta set n. The primary site sends this list to the snapshot copy facility at the secondary site. The secondary site retrieves the data of these blocks from the snapshot at the restart point, and returns the data to the primary site. The primary site receives these blocks of data and restores them to the primary file system.

In step 533, the snapshot copy facility starts replication to the primary file system, and creates an intermediate point snapshot (at state delta set n-x+δ), so all new changes made to the secondary file system since the intermediate point are kept and sent to the primary file system. However, they are not yet played back into the primary file system. At this point, the primary file system is mounted as a "raw" file system, so it is not accessible to users.

In step 534, the changes made to the secondary file system from the restart point to the intermediate point are copied from the secondary file system to the primary file system. These changes are maintained by the snapshot copy facility at the secondary site. For the snapshot copy facility as shown in FIG. 25, for example, assume that the restart point is the snapshot J and the intermediate point is the snapshot J+K where K is one. In this case, the changed blocks are identified in the bit map 507, and the data of the changed data blocks are found by reading from the intermediate point snapshot file system. To retrieve these changes, the snapshot copy facility scans the bit map 507 to find any set bit indicating a change in the save volume, and if the bit is set, the snapshot copy facility notes the block address (B_i). The snapshot copy facility then reads the snapshot file system (J+K) 483 at this block address (B_i). The snapshot copy facility continues this process until it has finished scanning the bit map 507.

In FIG. 28, after step 534, execution continues to step 535. In step 535, the primary file system is mounted as a read-only UxFS file system, and playback to the primary file system is started. At this point, the resync is finished, and the primary and secondary sites are ready for failback.

In step 531, if the delta set n-x+1 cannot be found in the save volume from before failover (459 in FIG. 29) at the primary site (for example, because the save volume was destroyed during the disaster), then execution branches from step 531 to step 536. In this case, the secondary file system is migrated to the primary file system, and a warning message is logged and displayed to the system administrator to signal that the full copy was required. In step 536, the secondary creates a first intermediate point snapshot (at state delta set n-x+δ), and copies the first intermediate point snapshot to the primary file

system. At this time, read/write access to the secondary file system is permitted on a priority basis. Then in step 537, the secondary starts replication to the primary file system, and creates a second intermediate point snapshot, so all new changes made to the secondary file system since the second intermediate point are kept and sent to the primary file system by the replication process. However, these new changes are not yet played back into the primary file system. At this point, the primary file system is mounted as a "raw" file system, so it is not accessible to users. Moreover, in step 537, the differential of the first and second intermediate point snapshots (i.e., the changes made to the secondary file system from the first intermediate point snapshot to the second intermediate point snapshot) are copied to the primary file system. Then execution continues to step 535, to mount the primary file system as a read-only UxFS file system, and to start playback to the primary file system. Also in step 535, the restart snapshot and the intermediate snapshot can be deleted.

The migration method of steps 536 to 537 is used in order to reduce the likelihood that read/write access to the secondary file system and replication to the primary file system would be interrupted by the primary and secondary save volumes (274, 275 in FIG. 5) becoming full. Each save volume functions as a wrap-around buffer. The primary save volume will store updates received in step 537 from the secondary site until playback to the primary file system is started in step 535. If the primary save volume would become full in step 537, a TCP flow control signal is sent from the primary site to the secondary site to suspend transmission of further updates. When transmission of further updates is suspended, the further updates can be stored in the secondary save volume (275 in FIG. 5) until the secondary save volume would become full. If the

secondary save volume would become full, then read/write access to the secondary file system and replication to the primary file system would be suspended until playback to the primary file system is started in step 535 and updates are removed from the primary and secondary save volumes.

FIG. 29 shows the state of the remote replication system of FIG. 20 during the resync procedure. At the secondary site, the snapshot copy facility 456 has created the intermediate point snapshot 457. A list of blocks in the delta sets n to n-x+1 is obtained from the save volume 459 at the primary site and sent to the snapshot copy facility 456 at the secondary site. The snapshot copy facility 456 returns "before images" of the requested blocks over an IP pipe 458 to the primary file system 451 to restore the primary file system to the state of the restart point snapshot, and then sends the changes from the restart point to the intermediate point. The replication service 452 at the secondary site is replicating changes to the secondary file system since the intermediate point, and is sending these changes over the IP pipe 460 to the playback service 450. Transmission of these changes since the intermediate point over the IP pipe 460 is concurrent with transmission of changes prior to the intermediate point over the IP pipe 458. Once the primary file system 451 has been synchronized to the intermediate point 457, the playback service 450 at the primary site is activated to play these changes into the primary file system 451. If the primary site should crash during the copy process of steps 534 or 536, then the copy process can be restarted automatically.

FIG. 30 is a flowchart of the subroutine to failback to the primary file system. In a first step 541, the primary file system is made almost identical to the secondary file system by the replication process. For example, when the primary file system is within

one delta chunk of the secondary file system, execution continues to step 542. In step 542, the secondary file system is re-mounted as read-only, and the last delta chunk is copied from the secondary file system to the primary file system in order to synchronize the primary file system from the secondary file system. Once synchronized, the primary and secondary are aborted, in order to terminate the processes of replication from the secondary, playback to the primary, and copying from the secondary to the primary. At this point, the primary file system contains all the data that the secondary file system had before the disaster and the additional data that the secondary file system had accumulated after the disaster. Then in step 543, there is a resumption of the replication of data from the primary file system and playback to the secondary file system. Then in step 544, the primary file system is remounted as read/write. Therefore, the recovery process permits replication to be restarted as it was before the disaster. Some data might have been lost at failover time since replication is asynchronous. This is dependent on the number "x" of delta sets that the secondary file system was behind the primary file system when the disaster occurred at the primary. The less the number "x" of delta sets the closer the final recovered file system would be to the original file system.

A number of commands have been devised for conducting the above recovery procedure when network file servers (such as shown in FIGS. 1 and 5) are used at the primary and secondary sites. These commands include a $fs_replicate-failover command, a $fs_replicate –resync command, a $fs_replicate–status command, $fs_replicate–failback command, and a $fs_copy –start command.

The $fs_replicate-failover command has the following format:

$fs_replicate –failover <pfs>:cel=<server> <sfs> [-sync] [–now]

where <pfs> is the primary file system, <server> is the server of the primary file system, <sfs> is the secondary file system, [-sync] is an option that is set when the user wants the failover to happen when both the primary file system and the secondary file system are "in sync" (i.e., identical state), and [-now] is an option to specify immediate failover. The $fs_replicate–failover command is executed on the control station for the secondary file system.

When the [-sync] option is not set, this command will internally stop the replication and playback of <sfs>, making sure that all of the delta sets available on secondary site are re-played. Then it will try to stop the replication on <pfs> as well, unless the [–restartpoint] option is set. Then <sfs> will be remounted "rw". If possible, <pfs> is remounted "ro". No [-sync] option should be specified if the primary control station or server is down. In this case, after failover, some data could be lost.

When the [now] option is set, the playback of <sfs> is immediately stopped, without replaying any of the delta sets available on the secondary site.

When the [-sync] option is specified, the primary control station and the server must be up, else the command will fail. The primary file system <pfs> is re-mounted as read-only and a last delta set is created. Once the last delta set is played back to the secondary, the replication process is stopped on the <sfs>/<pfs>. The secondary file system <sfs> now contains all the data that the <pfs> had before the failover. The secondary file system <sfs> is now mounted "rw".

FIG. 31 shows a flowchart of execution of the $fs_replicate–failover command. In a first step 551, if the sync option is not set, then execution branches to step 552 to perform a failover without sync procedure, as described below with reference to FIG. 32. After step 552, execution is finished. In step 551, if the sync option is set, then execution continues to step 553. In step 553, if the primary site is not operational, execution returns reporting a command failure. Otherwise, if the primary site is operational, execution continues to step 554, to perform a failover with sync procedure, as described below with reference to FIG. 33.

FIG. 32 shows a flowchart of the failover without sync procedure. In step 561, if possible, the primary file system is remounted as a read-only file system. Then in step 562, the secondary site finishes playback of the delta sets that it had received but had not played back prior to receipt of the failover command. In step 563, the secondary site creates a snapshot copy of the secondary file system. This snapshot is called the restarting point of the file system. In step 564, the secondary file system is remounted as a read/write file system, and execution of the failover command is finished.

FIG. 33 shows a flowchart of the failover with sync procedure. In a first step 571, the secondary file system is made almost identical to the primary file system by the replication process. For example, once the secondary file system is within one delta set chunk of the primary file system, execution continues to step 572. In step 572, the primary file system is remounted as read-only, the last delta chunk is copied from the primary file system to the secondary file system in order to synchronize the secondary file system to the primary file system, and then the primary and secondary are aborted to terminate the processes of replication from the primary, playback to the secondary, and

copying from the primary to the secondary. Then in step 573, the secondary site creates a snapshot copy of the secondary file system. This snapshot copy is called the restarting point of the file system. Finally, in step 574, the secondary file system is remounted as read/write.

The $fs_replicate–resync command has the following format:

$fs_replicate –resync sfs pfs:cel=server1

where sfs is the secondary file system (this is now the source file system), pfs is the primary file system (this is the file system to be restored, a raw file system mounted on a server which has its IP interfaces configured), server1 is the site where a disaster occurred, and <ckpt_name> is the restart point name.

The $fs_replicate–status command has the following format:

$fs_replicate –status <pfs> <sfs>:cel=<remote_server>

where <pfs> is the primary file system, <sfs> is the secondary file system, and <remote_server> is the server of the secondary site. This command can be used on the control station for the primary site, and also on the control station for the secondary site. Typically this command is used on the primary site when the user wants to check the status of the replication process, to determine when it is reasonable to failback. It is

considered reasonable to failback when the difference between the primary file system and the secondary file system is very small. To assist the user, the primary site responds to this command by providing an indication of the number of blocks that are different between the primary file system and the secondary file system.

Following is an example of the information provided when the $fs_replicate-status command is executed:

```
[nasadmin@cel_sfs nasadmin]$ fs_replicate -info sfs10
-v 128
id                      = 59
name                    = sfs10
type                    = replication
current_delta_set       = 16
current_block_number    = 1088
current_transfer_rate   = 17.0667 blocks/second
avg_transfer_rate       = 25.7444 blocks/second
id                      = 31
name                    = pfs10:cel_ip2
type                    = playback
current_delta_set       = 16

outstanding delta sets:
Delta  Source_create_time    Blocks
-----  ------------------    ------
15     11/26 20:11:56        1
```

	Source			Destination			
Delta	Create Time	Dur	Blocks	Playback Time	Dur	Blocks	Dsin Group
15	11/26 20:11:56	0	1				
14	11/26 20:01:56	0	1				
13	11/26 19:51:56	0	1	11/26 21:08:48	0	1	2
12	11/26 19:41:56	0	1				
11	11/26 19:31:56	0	1	11/26 20:48:48	0	1	2
10	11/26 19:21:56	0	1				
9	11/26 19:11:56	0	1	1 /26 20:28:28	0	1	2
8	11/26 18:52:40	0	1	11/26 20:08:09	0	1	1
7	11/26 18:42:40	0	1				
6	11/26 18:32:40	0	1	11/26 19:48:09	0	1	2
5	11/26 18:22:39	0	1				
4	11/26 18:12:39	0	1	11/26 19:28:08	0	1	2
3	11/26 18:02:39	0	4	11/26 19:07:23	0	4	1

The $fs_replicate–failback command has the following format:

$fs_replicate –failback <pfs> <sfs>:cel=<remote_server>

where <pfs> is the primary file system (a UxFS file system mounted as "ro"), and <sfs> is a secondary file system. This command is used when the user wants to failback to the

primary site (after re-build phase). No data is lost and the replication will continue but it will be reversed (the primary file system, which runs playback, will then run replication and vice-versa for the secondary file system). This command can also be used to change the direction of replication, in the absence of any failure or disruption.

The $fs_replicate–failback command is executed on the site that the replication service is running (i.e., the site having the file system that is mounted as read/write). Typically, the $fs_replicate–failback command is executed on the primary site after it has been rebuilt. Then the primary file system is a raw file system that is restored from the secondary file system. The secondary file system is re-mounted read write. Replication is stopped on the secondary and playback is stopped on the primary. After the primary file system and the secondary file systems are in sync, the primary file system is converted to a UxFS file system and re-mounted as read/write. Replication is turned back on and the flow of data is now from the primary file system to the secondary file system. Playback is started on the secondary file system. This command can be used to "swap" primary and secondary without having to do a full copy and without having to restart replication. On error, it should be possible to re-issue the fs_replicate -failback command and proceed if the error condition has been cleared.

The $fs_copy–start command has the following format:

$fs_copy –start <fs_ckpt2> <sfs>:cel=<remote_server> -fromfs <fs_ckpt1>

where <fs_ckpt1> is the primary file system ckpt1, <fs_ckpt2> is the primary file system ckpt2, <sfs> is the secondary file system, and <remote_server> is the server of the

secondary site. This command is used internally by the fs_replicate-resync command. It could also be used for other purposes, for example, as described below with reference to FIGS. 36-38. This command copies the delta from fs_ckpt1 to fs_ckpt2 over to the secondary file system, knowing that the state of the secondary file system was equal to fs_ckpt1.

The following is an example of how the commands can be used to recover from a disaster.

1. After the disaster, decide the option to use for failover.
2. $fs_replicate –failover pfs:cel=cel_pfs sfs
3. Verify that sfs can accept read/write operation.
4. Initialize replication from sfs to pfs:

   On PFS control station:

   server_mount server_2 –o ro pfs /pfs

   On SFS control station:

   $fs_replicate –resync pfs:cel=cel_pfs sfs
5. Verify that replication from sfs to pfs is running without a problem.
6. $fs_replicate –failback sfs pfs:cel=cel_pfs
7. Verify that replication from pfs to sfs is running without a problem.

Replication of Snapshots Using IP File System Copy Differential

As described above with reference to steps 534 and 537 of FIG. 28, a file system copy differential was used to replicate an intermediate point snapshot during recovery of

a file system from a disaster. A general-purpose file system copy differential command ($fs_copy–start) was also introduced.

FIG. 34 shows a flowchart of a procedure for the $fs_copy –start command. Here L is the index of the snapshot fs_ckpt1, and M is the index of the snapshot fs_ckpt2. In a first step 581, M is compared to L, and execution branches to step 582 to return an error if M is not greater than L. Otherwise, execution continues to step 583.

The following steps execute the $fs_copy –start command by scanning one or more of the bit maps of the snapshot copy facility of FIG. 25. Each bit map indicates whether or not each block of a respective snapshot changed from the snapshot to the next snapshot. Therefore, by examining the bit maps for the snapshots L to M-1, it is determined whether or not each block has changed from snapshot L to M. If so, the new value of the block is read from the snapshot M. The bit maps are scanned in block order so that the new blocks are read from the snapshot M of the primary (i.e., the source) file system and transmitted to the secondary (i.e., the destination) file system in the order of their block indices (B_i).

In step 583, the block index (B_i) is set to zero. In step 584, a snapshot index (I) is set to L. In step 585, the bit map for the snapshot (I) is indexed with the block index (B_i) to determine whether or not the block was changed between snapshot (I) and snapshot (I+1). If in step 585 the bit for the block (B_i) is not set in the bit map for the snapshot (I), then no such change occurred, and execution continues to step 586. In step 586, if the snapshot index I is not equal to M-1, then execution continues step 587. In step 587, the snapshot index I is incremented by one. After step 587, execution loops back to step 585.

In step 585, if the bit for the block (B_i) is set in the bit map for the snapshot (I),

then execution branches to step 589. In step 589, the snapshot (M) is read to get the new data for the block (B_i). Execution continues to step 590 of FIG. 35. In step 590, the block index (B_i) and the new data for the block (B_i) from the snapshot (M) are returned for copying into the secondary file system. Then in step 591, if the block index (B_i) is at the end of the production volume, then execution is finished. If not, execution loops back to step 588 of FIG. 34. In step 588, the block index (B_i) is incremented by one, and execution loops back to step 584, to continue scanning for the next value of the block index.

In step 586 of FIG. 34, if I is equal to M-1, then execution continues to step 591 of FIG. 35.

It should be understood that the flowchart of FIGS. 34-35 represents a program executed by at least one processor in a data storage system such as a network file server. The processor, for example, is a data mover computer (e.g., 232 in FIG. 1). The program, for example, is initially contained in a program storage device such as a floppy disk (e.g., 238 in FIG. 1) and down-loaded into storage of the data mover computer.

The program in the flowchart of FIGS. 34-35 has an inner loop including steps 585, 586, 587 that indexes the snapshots L to snapshot M-1. This sequence includes the snapshot L and the snapshots that are both younger than the snapshot L and older than the snapshot M. The program in the flowchart of FIGS. 34-35 has an outer loop including steps 584, 585, 586, 591, and 588 that indexes the blocks. When a bit in the indexed bit map is found to be set in step 585, the inner loop is exited to return the block index (B_i) and the data in the snapshot M for block (B_i).

The snapshot copy differential has been described above for facilitating recovery of a file system after a disaster. The snapshot copy differential can also be used for wide-area distribution of updates on an as-needed basis. This reduces network traffic for the case where a client has an old local version of a file system and needs a new version of the file system. A new local version of the file system can be constructed by copying the appropriate changes into the old local version of the file system.

FIG. 36, for example, shows a block diagram of a data network in which snapshot deltas are transmitted over a wide-area network 626 from a network file server 627 to a local file server 624 in order to update a local file system 625 as needed. The local file server 624 services local clients 621, 622, and 623. When a client needs a more recent version of the file system, and the local file system 625 is not the most recent version, then the local file server may request a specified version from the network file server 627, or the client may request the most recent version available.

The network file server 627 has a snapshot copy facility 628 storing multiple snapshots 629, 630. If the local file system 625 in the local file server 624 is one of the multiple snapshots, then the network file server 627 may respond to a request from the local file server 624 by obtaining from the snapshot copy facility 628 a snapshot copy differential that would contain all of the updates necessary to convert the local file system 624 to a more recent snapshot of the file system. In the usual case, the local file server 624 would request all of the updates necessary to convert the local file system 625 to the most recent snapshot copy. However, it is also possible for the local file server 624 to request the updates for a specified version that would not necessarily be the most recent snapshot copy.

FIGS. 37 and 38 show a flowchart of a procedure for the replication of the most recent snapshot in the system of FIG. 36 using the snapshot copy differential. In a first step 641, the client requests access to the file in the local file system in the local file server. In step 642, the local file server accesses attributes of the local file system and finds that it is version (Q) of a local updatable file system, last updated at time (Tu) from a network file server having a network identifier (NETID). The local file server uses the time of last update (Tu) to determine that it is time to check for a more recent version, and sends an update request to the network file server (NETID). For example, the file system has an attribute specifying a minimum update interval (Tmin), and it is not time to request an update unless the minimum update interval has elapsed since the time of the last update. The request specifies the version (Q) already in the local file server. In response, the network file server accesses the snapshot copy facility to find the oldest version (J) and the youngest version (J+K) stored in the network file server. In step 645, Q is compared to J+K. If Q=J+K, then execution branches to step 646 of FIG. 38. In step 646, the network file server returns a message that no more recent version is available. The local file server resets the time of last update (Tu) to the current time, and accesses the local version (Q) for the client. Upon completion of step 646, the snapshot replication process is finished.

If in step 645 Q is not equal to J+K, then execution continues to step 647 of FIG. 38. In step 647, Q is compared to K. If Q is less than K, then execution branches to step 648. In this case, the version (Q) is not a snapshot in the snapshot copy facility because the version (Q) is too old. In step 648, the network file server copies the youngest snapshot version (J+K) to the local file server. The local file server replaces the local

version (Q) with the new version (J+K), resets the time of last update (Tu) to the current time, and accesses the new local version (J+K) for the client. Upon completion of step 648, the snapshot replication process is finished.

In step 647, if Q is not less than K, then execution continues to step 649. In step 649, the network file server does a file system copy snapshot delta <Q> to <J+K> of blocks into the local version (Q) to convert it into the youngest snapshot version (J+K). The local file server resets the time of last update (Tu) to the current time, and accesses this local snapshot version (J+K) for the client. Upon completion of step 649, the snapshot replication process is finished.

In a preferred snapshot copy facility, as described below with reference to FIGS. 41 to 46, there is kept a meta bit map for each snapshot copy for indicating blocks of the production file system that are not used in the snapshot copy. Further details of such a snapshot copy facility are described in Philippe Armangau, et al., "Data Storage System Having Meta Bit Maps for Indicating Whether Data Blocks are Invalid in Snapshot Copies," U.S. Patent Application Ser. 10/213,241 filed Aug. 6, 2002, incorporated herein by reference. The snapshot copy facility maintains the meta bit maps in order to store the "before image" of a block in the save volume at the tail of the snapshot queue only when the block is being written to and the "before image" is needed for responding to any request for reading a snapshot copy. This reduces the number of blocks that are stored in the save volumes. However, in this case, the bit map for each snapshot (L) indicates whether or not a block has been stored in the save volume for the snapshot (L), and no longer will indicate all of the blocks that have been changed after snapshot (L) and before snapshot (L+1). In particular, if a block was not in use for snapshot (L), and was written

to after snapshot (L) and before snapshot (L+1), then the "before image" of the block will not be written into the save volume for snapshot (L).

When it is known that a block is not used in the snapshot copy (M), then there is no need for the snapshot copy facility to return the block when responding to a request for the snapshot delta of snapshot <L> to snapshot <M>. Therefore, for the preferred snapshot copy facility, it is desirable to modify the procedure of FIG. 34 in order use the information in the meta bit map for the snapshot <M>. In this case, the procedure of FIG. 34 should also be modified to account for the fact that the save volumes no longer store the "before images" for all of the blocks that may have changed between the successive snapshot copies.

FIG. 39 shows how the flowchart of FIG. 34 can be modified for use with the preferred snapshot copy facility of FIGS. 41 to 46. Steps 651, 652, and 653 of FIG. 39 are similar to steps 581, 582, and 583 of FIG. 34. Step 654 of FIG. 39 is similar to step 588 of FIG. 34, and step 658 of FIG. 39 is similar to step 589 of FIG. 34.

In FIG. 39, after step 653 or step 654, execution continues to step 655. In step 655, if the block (B_i) is not in use in the snapshot (M), then execution branches to step 591 of FIG. 35. Therefore, a block not in use in the snapshot (M) is not returned in response to the command to copy the snapshot delta <L> to <M>. If the block (B_i) is in use in the snapshot (M), then execution continues from step 655 to step 656. In step 656, if the block (B_i) is in any of the save volumes (L) to (M-1), then execution continues to step 658 to read the block (B_i) from the snapshot (M) in order to return this version of the block (B_i) in response to the command to copy the snapshot delta <L> to <M>. In step 656, if the block (B_i) is in any of the save volumes (L) to (M-1), then execution branches

to step 657. In step 657, if the block (B_i) is in use in all of the snapshots (L) to (M-1), then execution branches to step 591 of FIG. 35. In this case, the block (B_i) did not change from snapshot (L) to snapshot (M), because if it did, a "before image" would have been stored in one of the save volumes (L) to (M-1). In step 657 of FIG. 39, if it is not true that the block (B_i) is in use in all of the snapshots (i.e., it is true that the block (B_i) is not in use in at least one of the snapshots (L) to (M-1)), then execution continues from step 657 to step 658 to read the block (B_i) from the snapshot (M) in order to return this version of the block (B_i) in response to the command to copy the snapshot delta <L> to <M>. In this case, it is possible that the block changed from snapshot (L) to snapshot (M) despite the fact that the block (B_i) in not in any of the save volumes (L) to (M-1).

FIG. 40 shows a preferred implementation of the procedure of FIG. 39. Steps 661 to 665 of FIG. 40 are similar to steps 651 to step 655 of FIG. 39. In step 665, the meta bit map for snapshot (M) has a value for the block (B_i) indicating whether or not the block (B_i) is in use for the snapshot (M). In particular, a value of 1 indicates that the block (B_i) is in use for the snapshot (M). Steps 666, 669, and 670 of FIG. 40 are similar to step 584, 586 and 587 of FIG. 34. From step 666 or step 670, execution continues to step 667.

In step 667, if the bit map for snapshot (I) has a value of 1 for the block (B_i), then execution continues to step 671 to read the snapshot (M) to get data for the block (B_i), in order to return the data in response to the command to copy the snapshot delta <L> to <M>. In this case, the save volume for block (I) includes a "before image" for the block (B_i). Otherwise, if the bit map for snapshot (I) does not have a value of 1 for the block (B_i), execution branches from step 667 to step 668. In step 668, if the meta bit map for

the snapshot (I) does not have a value of 1 for the block (B_i), execution continues to step 671 to read the snapshot (M) to get data for the block (B_i), in order to return the data in response to the command to copy the snapshot delta <L> to <M>. In this case, the block (B_i) is not in use in the snapshot (I). Otherwise, if the meta bit map for the snapshot (I) has a value of 1 for the block (B_i), execution continues to step 669.

Maintenance of Meta Bit Maps in the Snapshot Copy Facility

In the above description of the snapshot copy process, and in particular with respect to FIG. 25, it was assumed that the original contents of a block of the production file system must be saved to the most recent save volume before the contents of the block are modified by a write access to the production file system. In practice, however, the original contents are often invalid, and therefore need not be saved. For example, many applications start with an empty file, and the file increases in size as data is written to the file. In some of these applications, the file rarely decreases in size. However, storage for the file may be released when the file is deleted from the file server, for example, when the file is transferred to archival storage. In some applications, the extent of a file may be dynamically decreased concurrent with read/write access to the file.

There are significant advantages to identifying when read/write access to the production file system is about to modify the contents of an invalid data block. If this can be done in an efficient manner, then there can be a decrease in the access time for write access to the production file system. A write operation to an invalid block can be executed immediately, without the delay of saving the original contents of the data block to the most recent save volume at the tail of the snapshot queue. Moreover, there is a

saving of storage because less storage is used for the save volumes. There is also a decrease in memory requirements and an increase in performance for the operations upon the snapshot file systems, because smaller bit and block hash indices can be used, and the reduced amount of storage for the snapshots can be more rapidly restored to the production file system, or deallocated for re-use when snapshots are deleted.

An efficient way of identifying when read/write access to the production file system is about to modify the contents of an invalid data block is to use a meta bit map having a bit for indicating whether or not each allocated block of storage in the production file system is valid or not. For example, whenever storage is allocated to the production file system by the initial allocation or the extension of a clone volume, a corresponding meta bit map is allocated or extended, and the bits in the meta bit map corresponding to the newly allocated storage are initially reset.

FIG. 41 shows a procedure for writing a specified block (Bi) to the production file system when there is a meta bit map for indicating invalid data blocks in the production file system. In a first step 681, a queue pointer is set to point to the queue entry at the tail of the snapshot queue. Next, in step 682, the bit map in this snapshot queue entry is accessed to test the bit for the specified block (Bi). Next, in step 683, if the tested bit is found to be set, then execution continues to step 684. In step 684, new data is written to the specified block (Bi) in the clone volume, and then execution returns.

In step 683, if the bit in the bit map is not set, then execution branches to step 685. In step 685, the meta bit map in the snapshot queue entry is accessed to test the bit for the specified block (Bi). Then, in step 686, execution continues to step 687 if this bit is found to be set. In step 687, the content of the block (Bi) is copied from the clone

volume to the next free block in the save volume at the tail of the snapshot queue. In step 688, an entry for the block (Bi) is inserted into the block map at the tail of the snapshot queue, and then the bit for the block (Bi) is set in the bit map at the tail of the snapshot queue. Execution continues from step 688 to step 684, to write new data to the block (Bi) in the clone volume.

In step 686, if the tested bit is found not to be set, then execution branches to step 689. If the queue pointer is pointing to the head of the queue, then execution branches to step 684, to write new data to the block (Bi) in the clone volume. Otherwise, if the queue pointer is not pointing to the head of the snapshot queue, then execution continues to step 690 to advance the queue pointer to the next snapshot queue entry toward the head of the snapshot queue. After step 690, execution loops back to step 682.

FIG. 42 shows an organization of the snapshots in the network file server when a respective meta bit map 692, 693 is maintained for each snapshot in addition to the meta bit map 691 for the production volume. It is desired to maintain a respective meta bit map for each snapshot so that whenever the production file system is restored with a snapshot file system, the meta bit map for the production file system can be restored with the meta bit map for each snapshot. For example, a meta bit map of the production file system is included in the production file system, so that whenever a snapshot copy of the production file system is created, a snapshot copy of the meta bit map is also created. Consequently, when the production file system is restored with a snapshot, the meta bit map of the production volume is replaced with the meta bit map of the snapshot.

As shown in FIG. 42, a meta bit map 691 is linked to the production file system 481 for indicating invalid blocks in the production file system. Each entry in the

snapshot queue 500 includes a respective meta bit map linked to the snapshot file system in the entry. For example, the queue entry (J+K) at the tail 501 of the queue has a meta bit map 692 linked to the snapshot file system 483, and the queue entry (J) at the head 502 of the queue includes a meta bit map 693 linked to the delta volume 505.

To reduce the memory and storage requirements for maintaining the bit maps 694, 696 and block maps 695, 697, the each bit map is organized as a set of pages indexed by a page table, and the each block map is organized as a set of hash lists indexed by a hash table. The bit maps and block maps 694, 695 at the queue entry (J+K) at the tail of the queue are initially created in a random access memory cache and written back to storage of the save volume 488 when a next snapshot of the production file system 481 is created. Thereafter the bit maps and block maps can be deallocated from the random access memory in order to free up cache memory, and later staged from storage to the cache memory when needed for read access to their respective snapshots.

FIG. 43 shows that the bit map 694 is organized as a page table 701 indexed by a set of most significant bits (MSB) of the block index (Bi), and pages 702 and 703 of the bit map linked to respective entries of the page table. The page table 701 includes a set of entries, each of which is either zero, indicating that the entire page is zero and therefore the page is not stored in storage or random access memory, or is a pointer to a page of the bit map.

FIG. 44 shows that the block map 695 is organized as a hash table 708 indexed by a hashing of the block index (Bi), and a set of hash lists 709 linked to respective entries of the hash table 708. Each non-zero entry in the hash table 708 points to a respective one of the hash lists 709. Each entry in each hash list includes a block address (B_i) to a

block in the clone volume, a corresponding block address (S_i) of the block in the save volume, and a value that is either zero indicating the end of the has list, or a pointer to the next entry in the list.

It is not necessary to retain the bit map 694 for the snapshot because the block map 695 can be accessed to provide the same information that is contained in the bit map 694. In particular, the block map 695 can be accessed simply to determine whether or not a specified block (Bi) is found in the block map, and therefore can be found in the save volume for the corresponding snapshot. However, the bit map 694 can be accessed more quickly to provide this information. Therefore, for applications such as backup where the snapshots would be accessed very infrequently, it may be desirable to conserve storage by discarding the bit map for the snapshot at the tail of the queue once the next snapshot is created. Otherwise, for an application such as described above with respect to FIGS. 36-38 where the snapshots would be accessed frequently, the bit map for the snapshot at the tail of the snapshot queue may be written to storage and maintained in a random access cache memory as needed.

FIG. 45 shows that the meta bit map 691 has a respective bit corresponding to each block in the clone volume, and in this example, each bit in the meta bit map corresponds to one and only one block in the clone volume. The meta bit map 691 includes a series of words, each with a multiple of M bits. In this example, a bit having a value of zero indicates a corresponding block that is invalid, and a bit having a value of one indicates a corresponding block that is valid.

The meta bit map, however, may have a granularity greater than one block per bit. For example, each bit in the meta bit map could indicate a range of block addresses,

which may include at least some valid data. The benefit to the increased granularity is a reduced size of the meta bit map at the expense of sometimes saving invalid data to the save volume. For example, FIG. 46 shows the interpretation of a meta bit map 691' having a granularity of two blocks per bit. Each bit is set if any one of the two corresponding blocks is valid, or conversely, each bit is clear only if neither of the two corresponding blocks is valid. In this case, the block address can be converted to a bit address by an integer division by two, for example, by an arithmetic right shift of the block address by one bit position.

In view of the above, there has been described a snapshot copy facility that stores a plurality of snapshot copies of a production file system. Each of the snapshot copies is a prior state of the production file system at a respective point in time. The snapshot copy facility receives a request for the difference between a specified older snapshot copy and a specified younger snapshot copy, and responds by returning the difference between the older snapshot copy and the younger snapshot copy. In a preferred implementation, the snapshot copy facility has an index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system. The indices are scanned for a sequence of the snapshot copies to determine the blocks that have changed, and the snapshot copy facility returns the block numbers and data in the younger snapshot copy for the blocks that have changed.

It has been shown how the difference between an older snapshot copy and a younger snapshot copy can be used for recovery purposes and also for distribution of

updates over a network on an as-needed basis. In general, a client has an older local copy of a snapshot and needs a more recent snapshot. The client requests a network file server to provide the difference between the older snapshot and the more recent snapshot. The client writes this difference into its older local copy to reconstruct the more recent snapshot.

What is claimed is:

1. A method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, said method comprising:

the snapshot copy facility receiving a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies; and

the snapshot copy facility responding to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

2. The method as claimed in claim 1, wherein the production file system includes blocks of data, and the snapshot copy facility returns an identification of each block that has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies, and the snapshot copy facility returns the data in the specified younger one of the snapshot copies for said each block that has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

3. The method as claimed in claim 2, wherein the identifications of the changed blocks and the data of the changed blocks are returned in a sequential block number order.

4. The method as claimed in claim 1, wherein the snapshot copy facility has an index for each snapshot copy for indicating changes between said each snapshot copy and a next snapshot copy of the production file system, and the method includes scanning the index for the specified older one of the snapshot copies.

5. The method as claimed in claim 4, wherein the index for at least one of the snapshot copies is a bit map.

6. The method as claimed in claim 4, wherein the index for at least one of the snapshot copies includes a hash table.

7. The method as claimed in claim 4, which includes scanning the indices for a sequence of the snapshot copies including the index for the specified older one of the snapshot copies and a respective index for each snapshot copy of the production file system that is both younger than the specified older one snapshot copies and older than the specified younger one of the snapshot copies.

8. The method as claimed in claim 7, wherein the indices for the sequence of the snapshot copies are scanned by a program routine having an outer loop indexing blocks of data in the file system, and an inner loop indexing the snapshot copies in the sequence of the snapshot copies.

9. The method as claimed in claim 1, wherein the snapshot copy facility has an index for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy, and the method includes scanning the index for the specified younger one of the snapshot copies, and when the index indicates that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

10. A method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, the snapshot copy facility having an index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system, wherein the method comprises:

scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

11. The method as claimed in claim 10, wherein at least one of the indices is a bit map.

12. The method as claimed in claim 10, wherein at least one of the indices includes a hash table.

13. The method as claimed in claim 10, which includes responding to a request for the difference between the older one of the snapshot copies and a younger one of the snapshot copies by:

returning a sequence of block numbers of the blocks that have changed between the older one of the snapshot copies and the younger one of the snapshot copies, and

returning the data in the younger one of the snapshot copies for the blocks that have changed between the older one of the snapshot copies and the younger one of the snapshot copies.

14. The method as claimed in claim 13, wherein the block numbers of the changed blocks and the data of the changed blocks are returned in a sequential block number order.

15. The method as claimed in claim 10, wherein the indices for the sequence of the snapshot copies are scanned by a program routine having an outer loop indexing respective blocks, and an inner loop indexing snapshot copies in the sequence of the snapshot copies.

16. The method as claimed in claim 15, wherein the snapshot copy facility has a meta bit map for each snapshot copy for indicating blocks of data that are known to be

invalid in said each snapshot copy, and the method includes scanning the meta bit map for the specified younger one of the snapshot copies, and when the meta bit map is found to indicate that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies by scanning the indices for the sequence of the snapshot copies.

17. A method of operating a snapshot copy facility that stores a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, the snapshot copy facility having a first index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system and that have a "before image" saved for said each snapshot copy, the snapshot copy facility having a second index for said each snapshot copy for indicating blocks of data that are not in use in said each snapshot copy; said method comprising:

responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by accessing the second index for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices for a sequence of the snapshot copies to determine blocks that have changed between an

older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

18. The method as claimed in claim 17, which also includes accessing at least one of the second indices for the snapshot copies in the sequence of the snapshot copies and finding that at least one of the blocks is not in use in at least one of the snapshot copies in the sequence of the snapshot copies to determine that said at least one of the blocks has changed between the older one of the snapshot copies and the younger one of the snapshot copies not changed.

19. A method of operating a network file server, the network file server having a snapshot copy facility for storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, said method comprising:

the network file server receiving a request for an update to a specified snapshot copy of the production file system;

the network file server responding to the request by checking whether the snapshot copy facility contains the specified snapshot copy of the production file system, and upon finding that the snapshot copy facility contains the specified snapshot copy of the production file system, the network file server returning the difference between the

specified snapshot copy of the production file system and a more recent snapshot copy of the production file system.

20. The network file server as claimed in claim 19, wherein the more recent snapshot copy of the production file system is the most recent one of the snapshot copies of the production file system that are stored in the snapshot copy facility.

21. The network file server as claimed in claim 19, wherein the request specifies the more recent snapshot copy of the production file system.

22. The network file server as claimed in claim 19, wherein the network file server returns the difference between the specified snapshot copy of the production file system and the more recent snapshot copy of the production file system by returning a series of block numbers for blocks of the production file system that have changed between the specified snapshot copy of the production file system and the more recent snapshot copy of the production file system, and returning the data in the more recent snapshot copy of the production file system for said each block that has changed between the specified one of the snapshot copies of the production file system and the more recent snapshot copy of the production file system.

23. In a data processing network having a client and a network file server, the network file server storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective

point in time, the client having a local version of an older one of the snapshot copies, a method of providing the client with a younger one of the snapshot copies, the method comprising:

the network file server determining the difference between the younger one of the snapshot copies and the older one of the snapshot copies;

the network file server transmitting the difference between the younger one of the snapshot copies and the older one of the snapshot copies to the local version of the older one of the snapshot copies; and

writing the difference between the younger one of the snapshot copies and the older one of the snapshot copies into the local version of the older one of the snapshot copies to produce a local version of the younger one of the snapshot copies.

24. The method as claimed in claim 23, wherein the network file server determines the difference between the younger one of the snapshot copies and the older one of the snapshot copies in response to an update request from the client, the update request specifying the older one of the snapshot copies.

25. The method as claimed in claim 23, wherein the network file server determines the difference between the younger one of the snapshot copies and the older one of the snapshot copies by determining blocks of the production file system that have changed between the younger one of the snapshot copies and the older one of the snapshot copies, and the network file server transmits the difference between the younger one of the snapshot copies and the older one of the snapshot copies to the local version of

the younger one of the snapshot copies by transmitting block identifiers for the blocks of the production file system that have changed between the younger one of the snapshot copies and the older one of the snapshot copies, and by transmitting the data in the younger one of the snapshot copies for the blocks of the production file system that have changed between the younger one of the snapshot copies and the older one of the snapshot copies.

26. A snapshot copy facility comprising:

storage for storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time; and

at least one processor programmed for receiving a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies; and for responding to the request by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

27. The snapshot copy facility as claimed in claim 26, wherein the production file system includes blocks of data, and said at least one processor is programmed for returning an identification of each block that has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies, and the snapshot copy facility returns the data in the specified younger one of the snapshot copies

for said each block that has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

28. The snapshot copy facility as claimed in claim 27, wherein said at least one processor is programmed to return the identifications of the changed blocks and the data of the changed blocks in a sequential block number order.

29. The snapshot copy facility as claimed in claim 26, wherein the snapshot copy facility has an index for each snapshot copy for indicating changes between said each snapshot copy and a next snapshot copy of the production file system, and said at least one processor is programmed for scanning the index for the specified older one of the snapshot copies.

30. The snapshot copy facility as claimed in claim 29, wherein the index for at least one of the snapshot copies is a bit map.

31. The snapshot copy facility as claimed in claim 29, wherein the index for at least one of the snapshot copies includes a hash table.

32. The snapshot copy facility as claimed in claim 29, wherein said at least one processor is programmed for scanning the indices for a sequence of the snapshot copies including the index for the specified older one of the snapshot copies and a respective index for each snapshot copy of the production file system that is both younger

than the specified older one snapshot copies and older than the specified younger one of the snapshot copies.

33. The snapshot copy facility as claimed in claim 32, wherein said at least one processor is programmed for scanning the indices for the sequence of the snapshot copies by a program routine having an outer loop indexing the blocks, and an inner loop indexing the snapshot copies in the sequence of the snapshot copies.

34. The snapshot copy facility as claimed in claim 26, wherein the snapshot copy facility has an index for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy, and said at least one processor is programmed for scanning the index for the specified younger one of the snapshot copies, and when the index indicates that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

35. A snapshot copy facility comprising:

storage for storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time;

an index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system, and

at least one processor programmed for scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

36. The snapshot copy facility as claimed in claim 35, wherein at least one of the indices is a bit map.

37. The snapshot copy facility as claimed in claim 35, wherein at least one of the indices includes a hash table.

38. The snapshot copy facility as claimed in claim 35, wherein the production file system includes blocks of data, and said at least one processor is programmed to respond to a request for the difference between the older one of the snapshot copies and a younger one of the snapshot copies by:

returning a sequence of block numbers of the blocks that have changed between the older one of the snapshot copies and the younger one of the snapshot copies, and

returning the data in the younger one of the snapshot copies for the blocks that have changed between the older one of the snapshot copies and the younger one of the snapshot copies.

39. The snapshot copy facility as claimed in claim 38, wherein said at least one processor is programmed to return the block numbers of the changed blocks and the data of the changed blocks in a sequential block number order.

40. The snapshot copy facility as claimed in claim 35, wherein said at least one processor is programmed for scanning the indices for the sequence of the snapshot copies by a program routine having an outer loop indexing the blocks, and an inner loop indexing the snapshot copies in the sequence of the snapshot copies.

41. The snapshot copy facility as claimed in claim 35, which includes a meta bit map for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy, and wherein said at least one processor is programmed for scanning the meta bit map for the specified younger one of the snapshot copies, and when the meta bit map is found to indicate that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies by scanning the indices for the sequence of the snapshot copies.

42. A snapshot copy facility comprising:

storage for storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time;

a first index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system and that have a "before image" for said each snapshot copy stored in the storage,

a second index for each snapshot copy for indicating blocks of data that are not in use in said each snapshot copy, and

at least one processor programmed for responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by accessing the second index for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices for a sequence of the snapshot copies to determine blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

43. The snapshot copy facility as claimed in claim 42, wherein said at least one processor is also programmed for accessing at least one of the second indices for the snapshot copies in the sequence of the snapshot copies and finding that at least one of the blocks is not in use in at least one of the snapshot copies in the sequence of the snapshot

copies to determine that said at least one of the blocks has changed between the older one of the snapshot copies and the younger one of the snapshot copies not changed.

44. A network file server comprising a snapshot copy facility for storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time,

wherein the network file server is programmed for receiving a request for an update to a specified snapshot copy of the production file system, and responding to the request by checking whether the snapshot copy facility contains the specified snapshot copy of the production file system, and upon finding that the snapshot copy facility contains the specified snapshot copy of the production file system, returning the difference between the specified snapshot copy of the production file system and a more recent snapshot copy of the production file system.

45. The network file server as claimed in claim 44, wherein the more recent snapshot copy of the production file system is the most recent one of the snapshot copies of the production file system that are stored in the snapshot copy facility.

46. The network file server as claimed in claim 44, wherein the request specifies the more recent snapshot copy of the production file system.

47. The network file server as claimed in claim 44, wherein the network file server is programmed to return the difference between the specified snapshot copy of the

production file system and the more recent snapshot copy of the production file system by returning a series of block numbers for blocks of the production file system that have changed between the specified snapshot copy of the production file system and the more recent snapshot copy of the production file system, and the data in the more recent snapshot copy of the production file system for said each block that has changed between the specified one of the snapshot copies of the production file system and the more recent snapshot copy of the production file system.

48. The network file server as claimed in claim 44, wherein the network file server is programmed to return the more recent snapshot copy of the production file system upon finding that the snapshot copy facility does not contain the specified snapshot copy of the production file system.

49. A program storage device containing a program for a snapshot copy facility, the snapshot copy facility storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, wherein the program is executable for responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by returning the difference between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

50. The program storage device as claimed in claim 49, wherein the program is executable for returning an identification of each block that has changed between the

specified older one of the snapshot copies and the specified younger one of the snapshot copies, and for returning the data in the specified younger one of the snapshot copies for said each block that has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies.

51. The program storage device as claimed in claim 50, wherein the program is executable for returning the identifications of the changed blocks and the data of the changed blocks in a sequential block number order.

52. The program storage device as claimed in claim 49, wherein the snapshot copy facility has an index for each snapshot copy for indicating changes between said each snapshot copy and a next snapshot copy of the production file system, and the program is executable for scanning the index for the specified older one of the snapshot copies.

53. The program storage device as claimed in claim 52, wherein the program is executable for scanning the indices for a sequence of the snapshot copies including the index for the specified older one of the snapshot copies and a respective index for each snapshot copy of the production file system that is both younger than the specified older one snapshot copies and older than the specified younger one of the snapshot copies.

54. The program storage device as claimed in claim 53, wherein the program is executable for scanning the indices for the sequence of the snapshot copies by a

program routine having an outer loop indexing the blocks, and an inner loop indexing the snapshot copies in the sequence of the snapshot copies.

55. A program storage device containing a program for a snapshot copy facility, the snapshot copy facility having a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, and an index for each snapshot copy for indicating blocks of data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system, wherein the program is executable for scanning the indices for a sequence of the snapshot copies to determine the blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

56. The program storage device as claimed in claim 55, wherein the program is executable for responding to a request for the difference between the older one of the snapshot copies and a younger one of the snapshot copies by:

returning a sequence of block numbers of the blocks that have changed between the older one of the snapshot copies and the younger one of the snapshot copies, and

returning the data in the younger one of the snapshot copies for the blocks that have changed between the older one of the snapshot copies and the younger one of the snapshot copies.

57. The program storage device as claimed in claim 56, wherein the program is executable for returning the block numbers of the changed blocks and the data of the changed blocks in a sequential block number order.

58. The program storage device as claimed in claim 55, wherein the program is executable for scanning the indices for the sequence of the snapshot copies by a program routine having an outer loop indexing the blocks, and an inner loop indexing the snapshot copies in the sequence of the snapshot copies.

59. The program storage device as claimed in claim 55, wherein the snapshot copy facility has a meta bit map for each snapshot copy for indicating blocks of data that are known to be invalid in said each snapshot copy, and wherein the program storage device is executable for scanning the meta bit map for the specified younger one of the snapshot copies, and when the meta bit map is found to indicate that a block is not known to be invalid, then determining whether the block has changed between the specified older one of the snapshot copies and the specified younger one of the snapshot copies by scanning the indices for the sequence of the snapshot copies.

60. A program storage device containing a program for a snapshot copy facility, the snapshot copy facility having a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time, a first index for each snapshot copy for indicating blocks of

data in the production file system that have changed between said each snapshot copy and a next snapshot copy of the production file system and that have a "before image" for said each snapshot copy stored in the snapshot copy facility, and a second index for each snapshot copy for indicating blocks of data that are not in use in said each snapshot copy, wherein the program is executable for responding to a request for the difference between a specified older one of the snapshot copies and a specified younger one of the snapshot copies by accessing the second index for the specified younger one of the snapshot copies to determine blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, and for blocks of data in the production file system that are in use in the specified younger one of the snapshot copies, accessing at least one of the first indices for a sequence of the snapshot copies to determine blocks that have changed between an older one of the snapshot copies and a younger one of the snapshot copies, the sequence of the snapshot copies including the older one of the snapshot copies and each of the snapshot copies that is both younger than the older one of the snapshot copies and older than the younger one of the snapshot copies.

61. The program storage device as claimed in claim 60, wherein the program is executable for accessing at least one of the second indices for the snapshot copies in the sequence of the snapshot copies and finding that at least one of the blocks is not in use in at least one of the snapshot copies in the sequence of the snapshot copies to determine that said at least one of the blocks has changed between the older one of the snapshot copies and the younger one of the snapshot copies not changed.

62. A program storage device containing a program for a network file server, the network file server including a snapshot copy facility for storing a plurality of snapshot copies of a production file system, each of the snapshot copies being a prior state of the production file system at a respective point in time,

wherein the program is executable for receiving a request for an update to a specified snapshot copy of the production file system, and responding to the request by checking whether the snapshot copy facility contains the specified snapshot copy of the production file system, and upon finding that the snapshot copy facility contains the specified snapshot copy of the production file system, returning the difference between the specified snapshot copy of the production file system and a more recent snapshot copy of the production file system.

63. The program storage device as claimed in claim 62, wherein the more recent snapshot copy of the production file system is the most recent one of the snapshot copies of the production file system that are stored in the snapshot copy facility.

64. The program storage device as claimed in claim 62, wherein the request specifies the more recent snapshot copy of the production file system.

65. The program storage device as claimed in claim 62, wherein the program is executable for returning the difference between the specified snapshot copy of the production file system and the more recent snapshot copy of the production file system

by returning a series of block numbers for blocks of the production file system that have changed between the specified snapshot copy of the production file system and the more recent snapshot copy of the production file system, and returning the data in the more recent snapshot copy of the production file system for said each block that has changed between the specified one of the snapshot copies of the production file system and the more recent snapshot copy of the production file system.

66. The program storage device as claimed in claim 62, wherein the program is executable for returning the more recent snapshot copy of the production file system upon finding that the snapshot copy facility does not contain the specified snapshot copy of the production file system.

ABSTRACT

A snapshot copy facility stores snapshot copies of a production file system. The snapshot copy facility receives a request for the difference between a specified older snapshot copy and a specified younger snapshot copy, and responds by returning the difference between the older snapshot copy and the younger snapshot copy. In a preferred implementation, the snapshot copy facility has an index for each snapshot copy for indicating blocks of data in the production file system that have changed between the snapshot copy and a next snapshot copy of the production file system. The indices are scanned for a sequence of the snapshot copies to determine the blocks that have changed, and the snapshot copy facility returns the block numbers and data in the younger snapshot copy for the blocks that have changed.

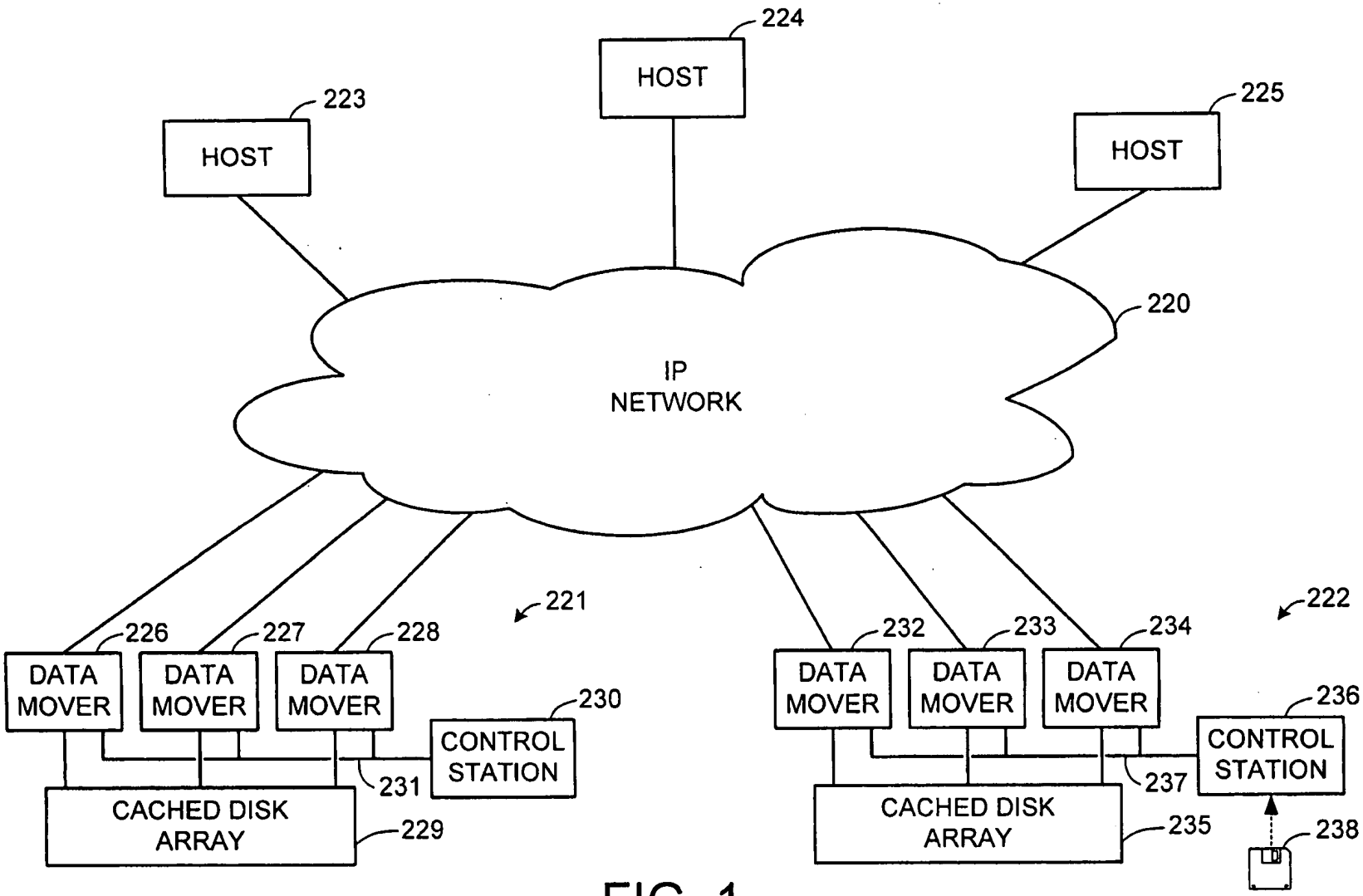

224
HOST
223
HOST
225
HOST
220
IP
NETWORK
221
222
226
227
228
232
233
234
DATA
MOVER
DATA
MOVER
DATA
MOVER
DATA
MOVER
DATA
MOVER
DATA
MOVER
230
CONTROL
STATION
231
CACHED DISK
ARRAY
229
236
CONTROL
STATION
237
CACHED DISK
ARRAY
235
238

FIG. 1

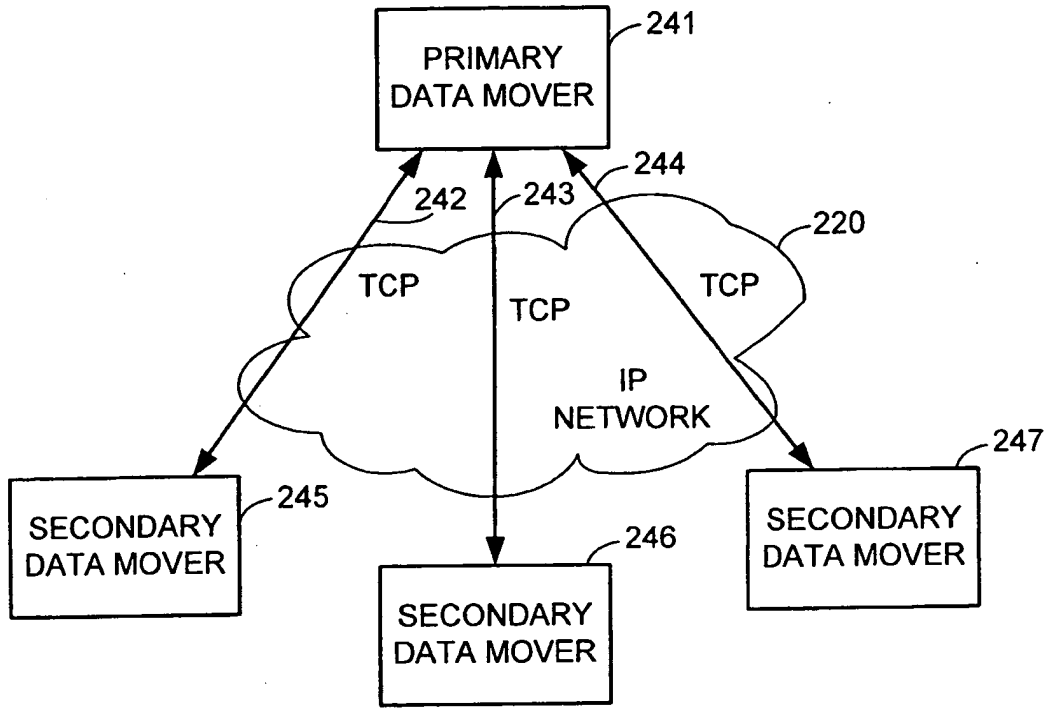

PRIMARY
DATA MOVER
241
242
243
244
220
TCP
TCP
TCP
IP
NETWORK
SECONDARY
DATA MOVER
245
SECONDARY
DATA MOVER
246
SECONDARY
DATA MOVER
247

FIG. 2

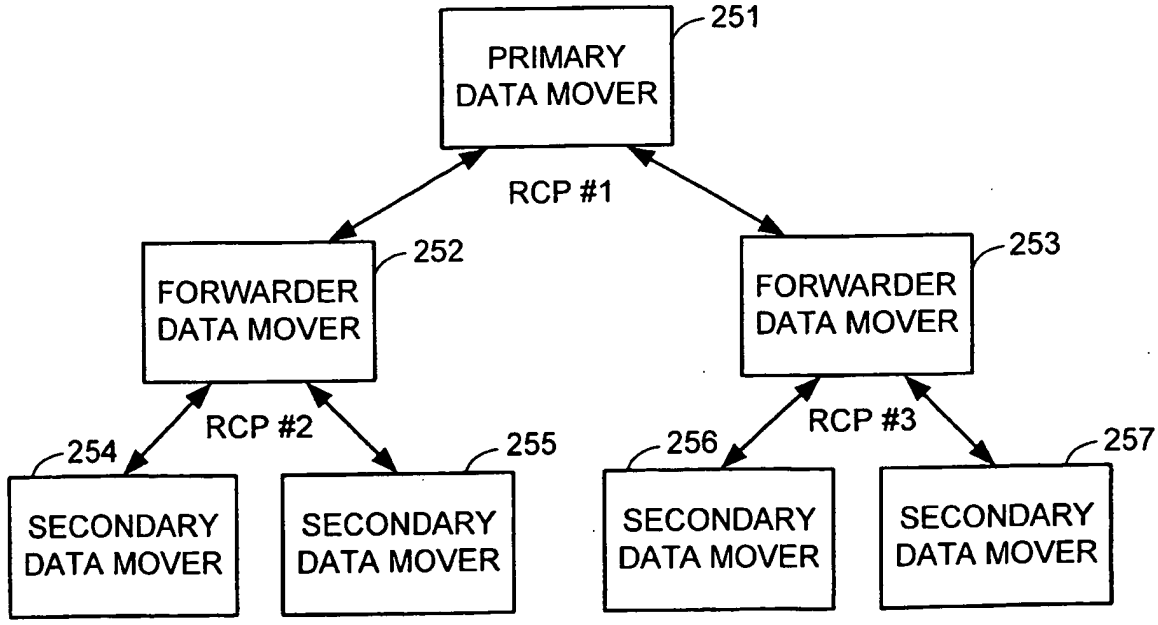

PRIMARY
DATA MOVER
251
RCP #1
FORWARDER
DATA MOVER
252
FORWARDER
DATA MOVER
253
RCP #2
RCP #3
254
255
256
257
SECONDARY
DATA MOVER
SECONDARY
DATA MOVER
SECONDARY
DATA MOVER
SECONDARY
DATA MOVER

FIG. 3

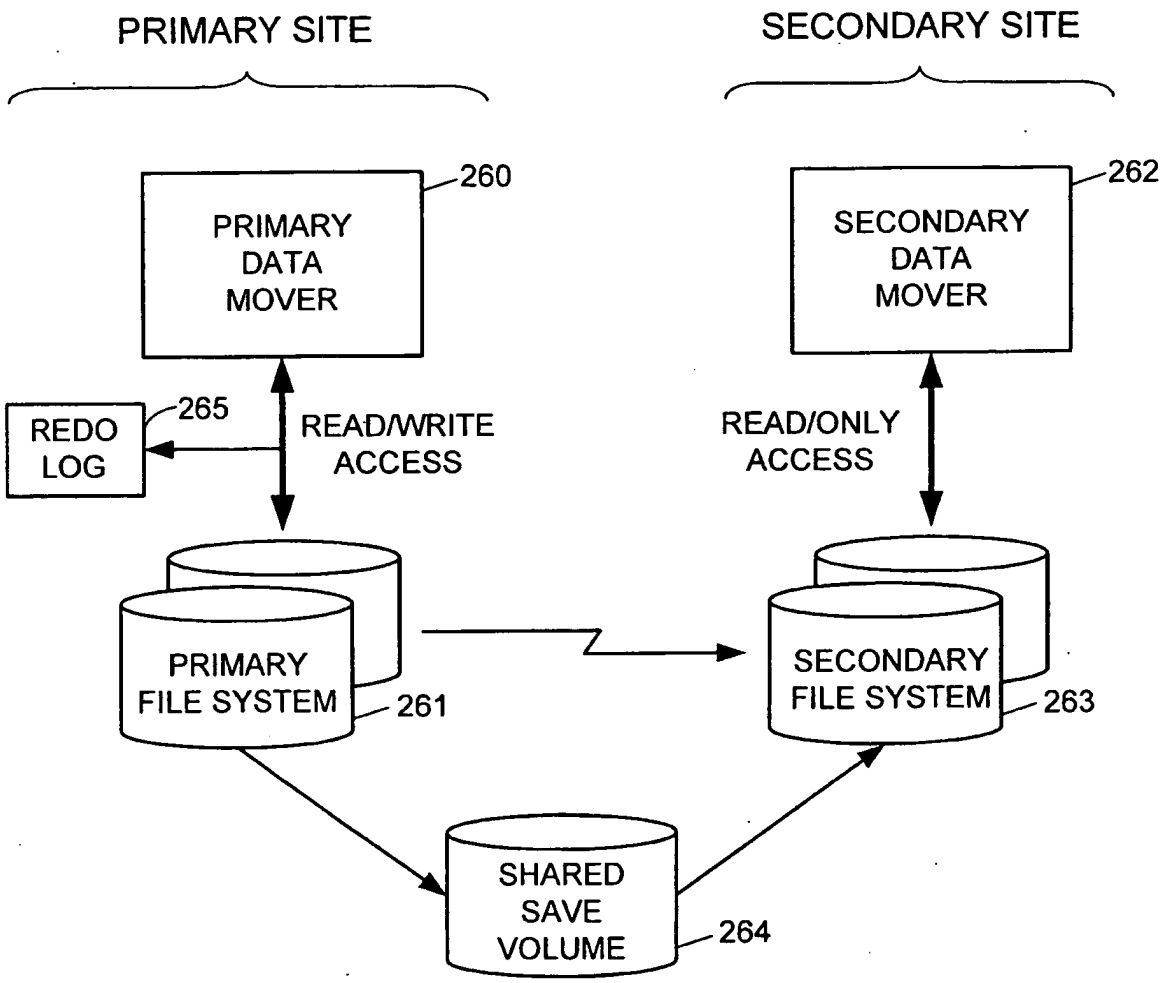

PRIMARY SITE
SECONDARY SITE
PRIMARY DATA MOVER
260
SECONDARY DATA MOVER
262
REDO LOG
265
READ/WRITE ACCESS
READ/ONLY ACCESS
PRIMARY FILE SYSTEM
261
SECONDARY FILE SYSTEM
263
SHARED SAVE VOLUME
264

FIG. 4

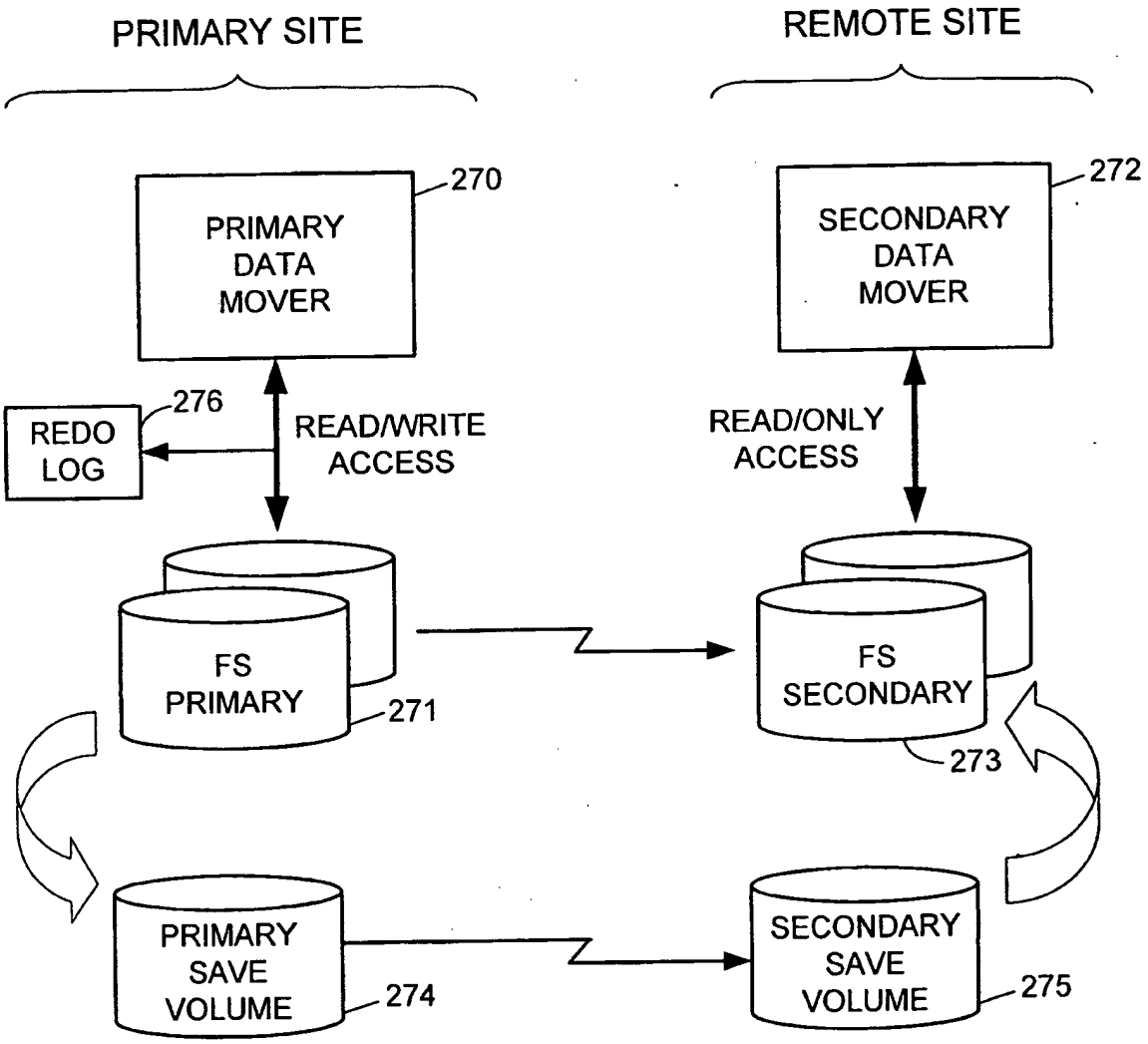

PRIMARY SITE
REMOTE SITE
PRIMARY DATA MOVER
270
SECONDARY DATA MOVER
272
REDO LOG
276
READ/WRITE ACCESS
READ/ONLY ACCESS
FS PRIMARY
271
FS SECONDARY
273
PRIMARY SAVE VOLUME
274
SECONDARY SAVE VOLUME
275

FIG. 5

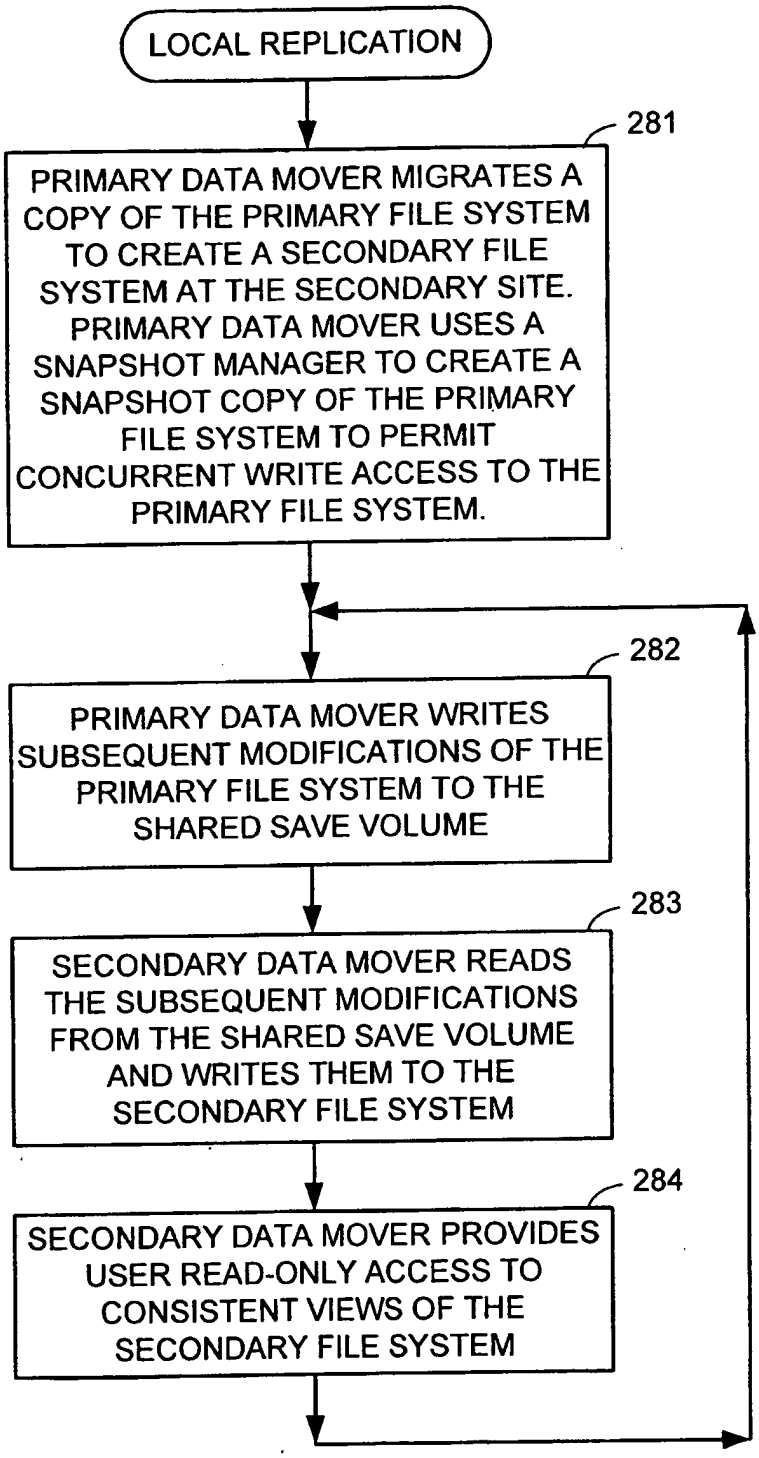

LOCAL REPLICATION
281
PRIMARY DATA MOVER MIGRATES A COPY OF THE PRIMARY FILE SYSTEM TO CREATE A SECONDARY FILE SYSTEM AT THE SECONDARY SITE. PRIMARY DATA MOVER USES A SNAPSHOT MANAGER TO CREATE A SNAPSHOT COPY OF THE PRIMARY FILE SYSTEM TO PERMIT CONCURRENT WRITE ACCESS TO THE PRIMARY FILE SYSTEM.
282
PRIMARY DATA MOVER WRITES SUBSEQUENT MODIFICATIONS OF THE PRIMARY FILE SYSTEM TO THE SHARED SAVE VOLUME
283
SECONDARY DATA MOVER READS THE SUBSEQUENT MODIFICATIONS FROM THE SHARED SAVE VOLUME AND WRITES THEM TO THE SECONDARY FILE SYSTEM
284
SECONDARY DATA MOVER PROVIDES USER READ-ONLY ACCESS TO CONSISTENT VIEWS OF THE SECONDARY FILE SYSTEM

FIG. 6

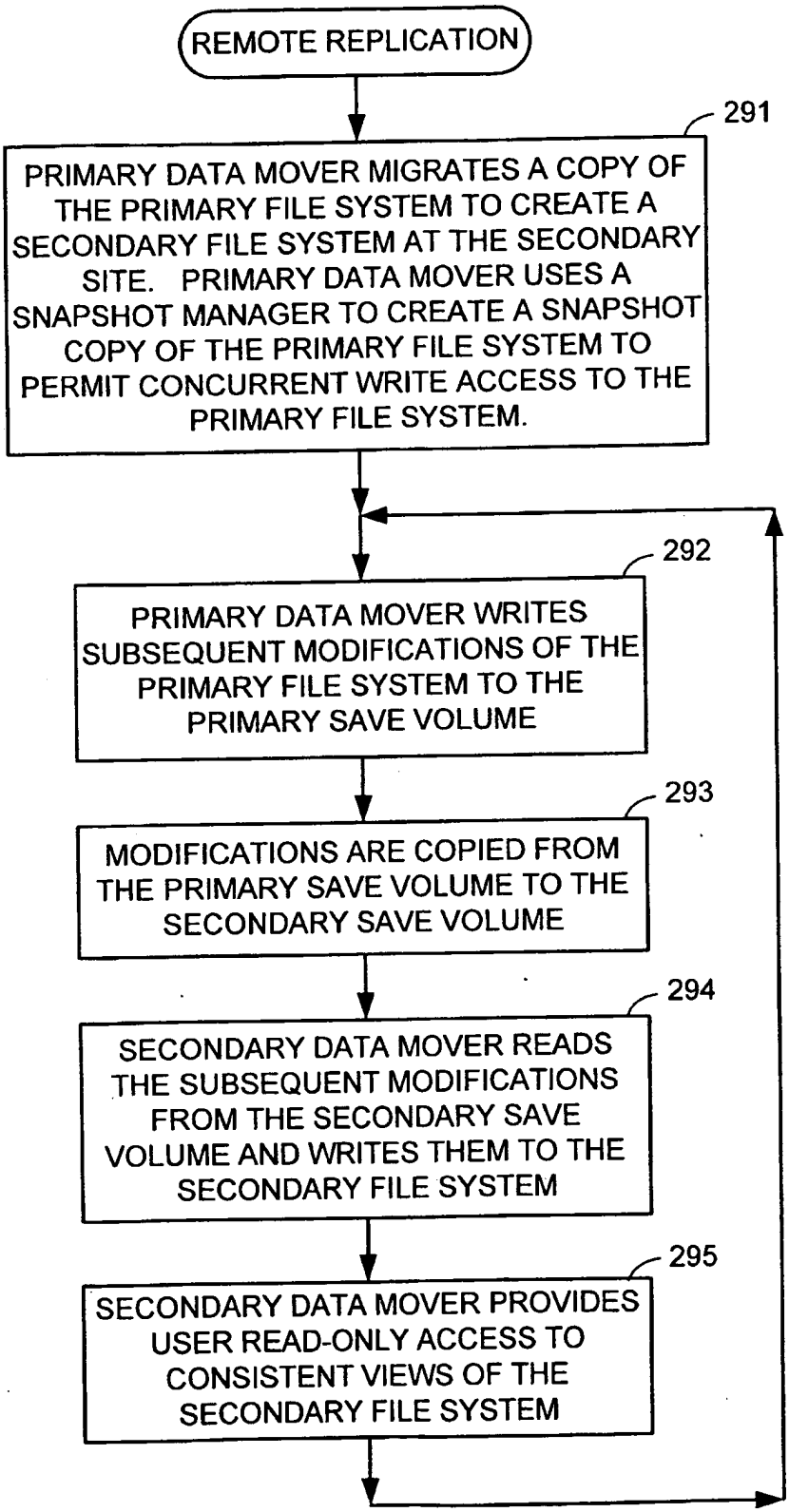

REMOTE REPLICATION
291
PRIMARY DATA MOVER MIGRATES A COPY OF THE PRIMARY FILE SYSTEM TO CREATE A SECONDARY FILE SYSTEM AT THE SECONDARY SITE. PRIMARY DATA MOVER USES A SNAPSHOT MANAGER TO CREATE A SNAPSHOT COPY OF THE PRIMARY FILE SYSTEM TO PERMIT CONCURRENT WRITE ACCESS TO THE PRIMARY FILE SYSTEM.
292
PRIMARY DATA MOVER WRITES SUBSEQUENT MODIFICATIONS OF THE PRIMARY FILE SYSTEM TO THE PRIMARY SAVE VOLUME
293
MODIFICATIONS ARE COPIED FROM THE PRIMARY SAVE VOLUME TO THE SECONDARY SAVE VOLUME
294
SECONDARY DATA MOVER READS THE SUBSEQUENT MODIFICATIONS FROM THE SECONDARY SAVE VOLUME AND WRITES THEM TO THE SECONDARY FILE SYSTEM
295
SECONDARY DATA MOVER PROVIDES USER READ-ONLY ACCESS TO CONSISTENT VIEWS OF THE SECONDARY FILE SYSTEM

FIG. 7

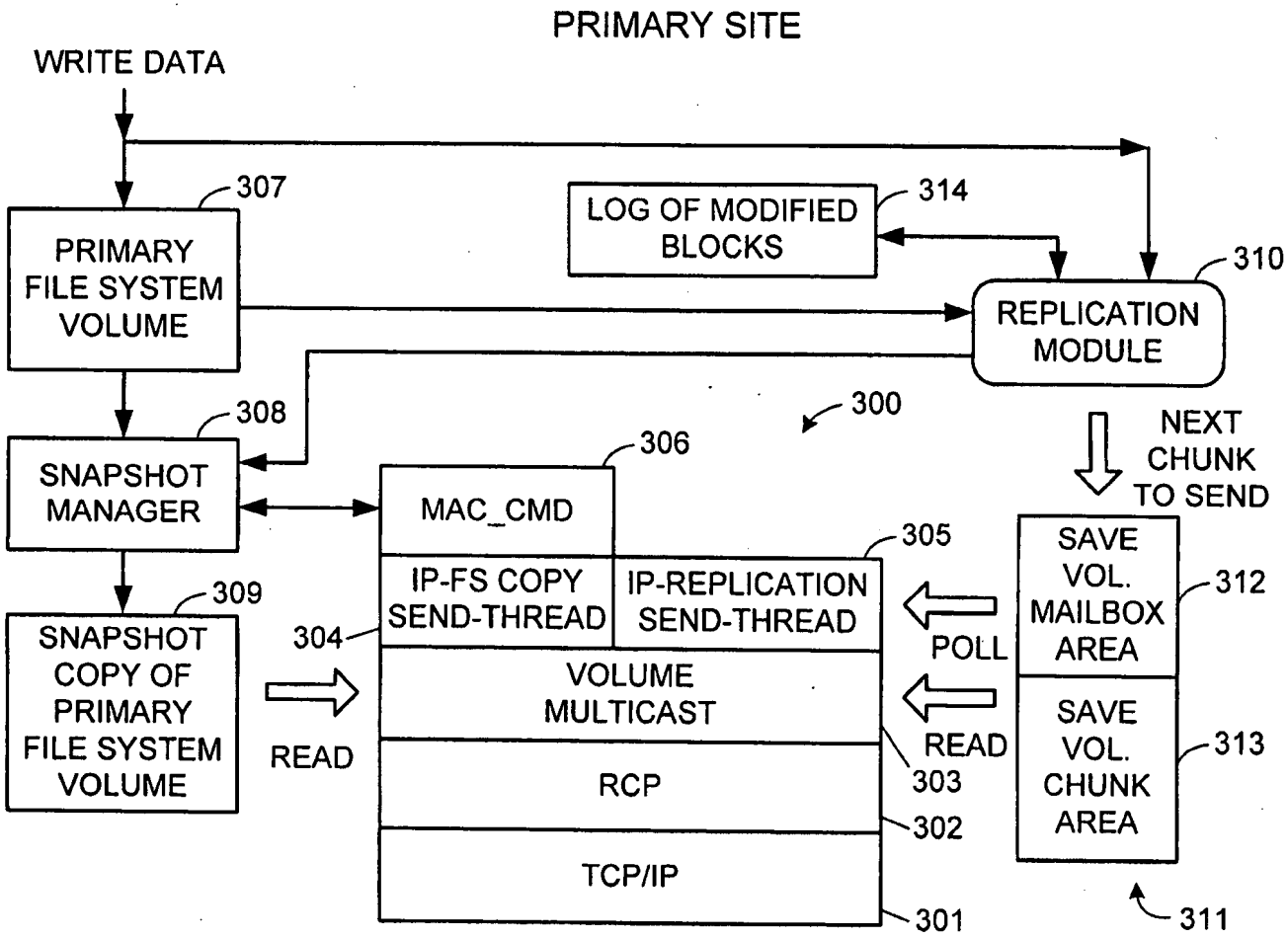

PRIMARY SITE
WRITE DATA
307
PRIMARY FILE SYSTEM VOLUME
LOG OF MODIFIED BLOCKS
314
310
REPLICATION MODULE
300
308
SNAPSHOT MANAGER
306
MAC_CMD
NEXT CHUNK TO SEND
305
IP-FS COPY SEND-THREAD
IP-REPLICATION SEND-THREAD
SAVE VOL. MAILBOX AREA
312
309
SNAPSHOT COPY OF PRIMARY FILE SYSTEM VOLUME
304
POLL
VOLUME MULTICAST
READ
READ
SAVE VOL. CHUNK AREA
313
303
RCP
302
TCP/IP
301
311

FIG. 8

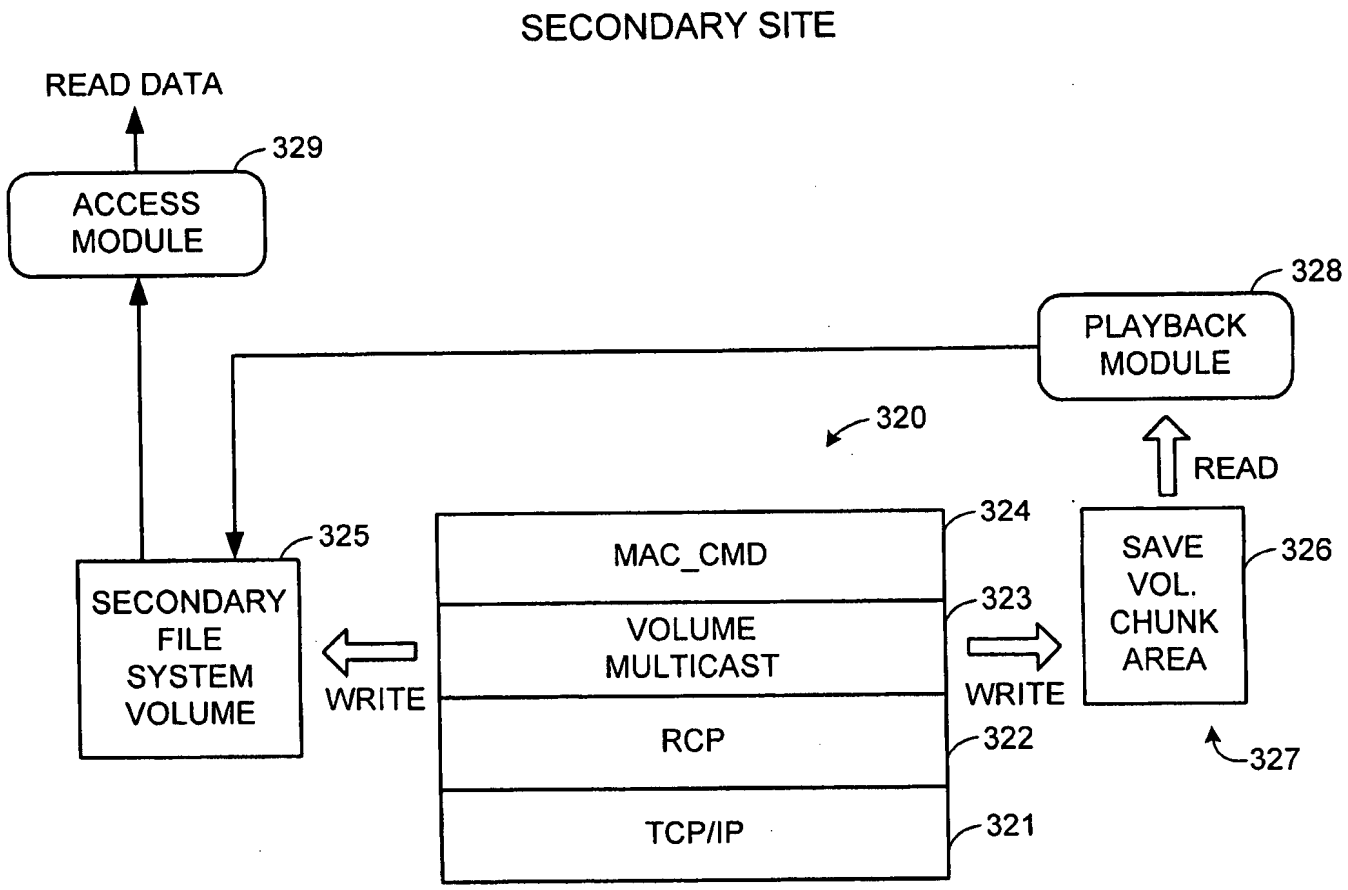

SECONDARY SITE
READ DATA
329
ACCESS MODULE
328
PLAYBACK MODULE
320
READ
325
SECONDARY FILE SYSTEM VOLUME
WRITE
MAC_CMD
VOLUME MULTICAST
RCP
TCP/IP
324
323
322
321
WRITE
SAVE VOL. CHUNK AREA
326
327

FIG. 9

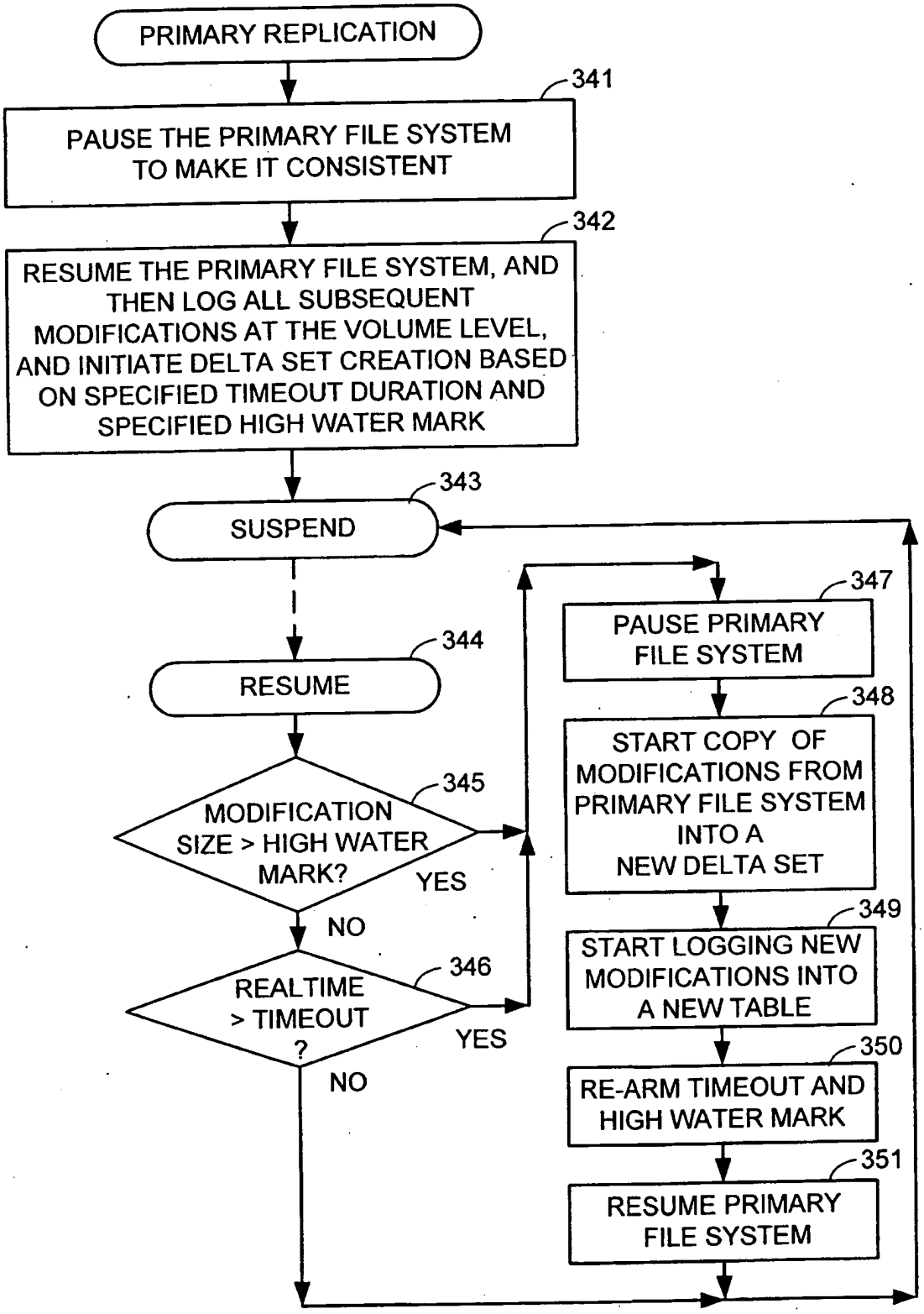

PRIMARY REPLICATION
341
PAUSE THE PRIMARY FILE SYSTEM TO MAKE IT CONSISTENT
342
RESUME THE PRIMARY FILE SYSTEM, AND THEN LOG ALL SUBSEQUENT MODIFICATIONS AT THE VOLUME LEVEL, AND INITIATE DELTA SET CREATION BASED ON SPECIFIED TIMEOUT DURATION AND SPECIFIED HIGH WATER MARK
343
SUSPEND
344
RESUME
345
MODIFICATION SIZE > HIGH WATER MARK?
YES
NO
346
REALTIME > TIMEOUT ?
YES
NO
347
PAUSE PRIMARY FILE SYSTEM
348
START COPY OF MODIFICATIONS FROM PRIMARY FILE SYSTEM INTO A NEW DELTA SET
349
START LOGGING NEW MODIFICATIONS INTO A NEW TABLE
350
RE-ARM TIMEOUT AND HIGH WATER MARK
351
RESUME PRIMARY FILE SYSTEM

FIG. 10

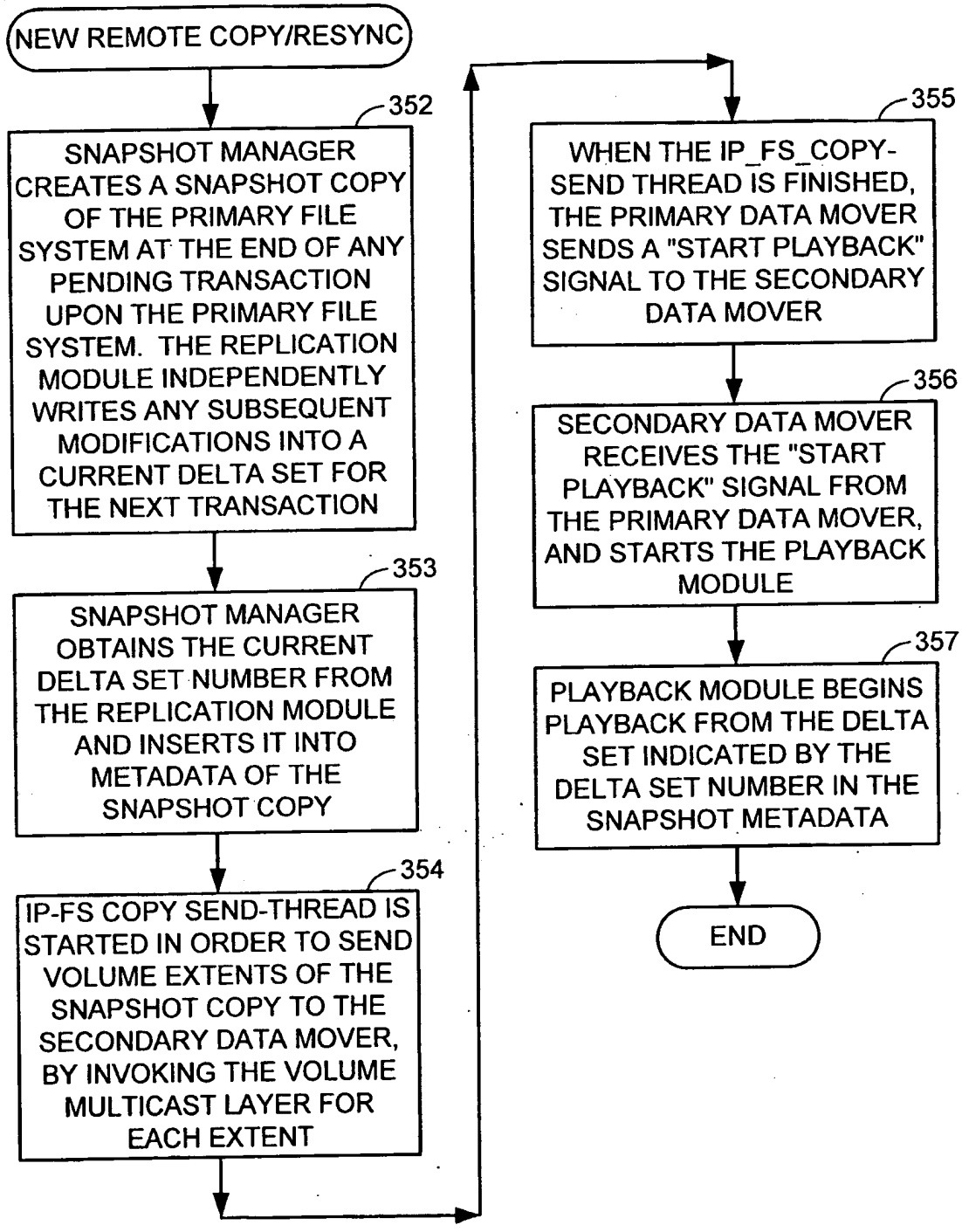

NEW REMOTE COPY/RESYNC
352
SNAPSHOT MANAGER CREATES A SNAPSHOT COPY OF THE PRIMARY FILE SYSTEM AT THE END OF ANY PENDING TRANSACTION UPON THE PRIMARY FILE SYSTEM. THE REPLICATION MODULE INDEPENDENTLY WRITES ANY SUBSEQUENT MODIFICATIONS INTO A CURRENT DELTA SET FOR THE NEXT TRANSACTION
353
SNAPSHOT MANAGER OBTAINS THE CURRENT DELTA SET NUMBER FROM THE REPLICATION MODULE AND INSERTS IT INTO METADATA OF THE SNAPSHOT COPY
354
IP-FS COPY SEND-THREAD IS STARTED IN ORDER TO SEND VOLUME EXTENTS OF THE SNAPSHOT COPY TO THE SECONDARY DATA MOVER, BY INVOKING THE VOLUME MULTICAST LAYER FOR EACH EXTENT
355
WHEN THE IP_FS_COPY-SEND THREAD IS FINISHED, THE PRIMARY DATA MOVER SENDS A "START PLAYBACK" SIGNAL TO THE SECONDARY DATA MOVER
356
SECONDARY DATA MOVER RECEIVES THE "START PLAYBACK" SIGNAL FROM THE PRIMARY DATA MOVER, AND STARTS THE PLAYBACK MODULE
357
PLAYBACK MODULE BEGINS PLAYBACK FROM THE DELTA SET INDICATED BY THE DELTA SET NUMBER IN THE SNAPSHOT METADATA
END

FIG. 11

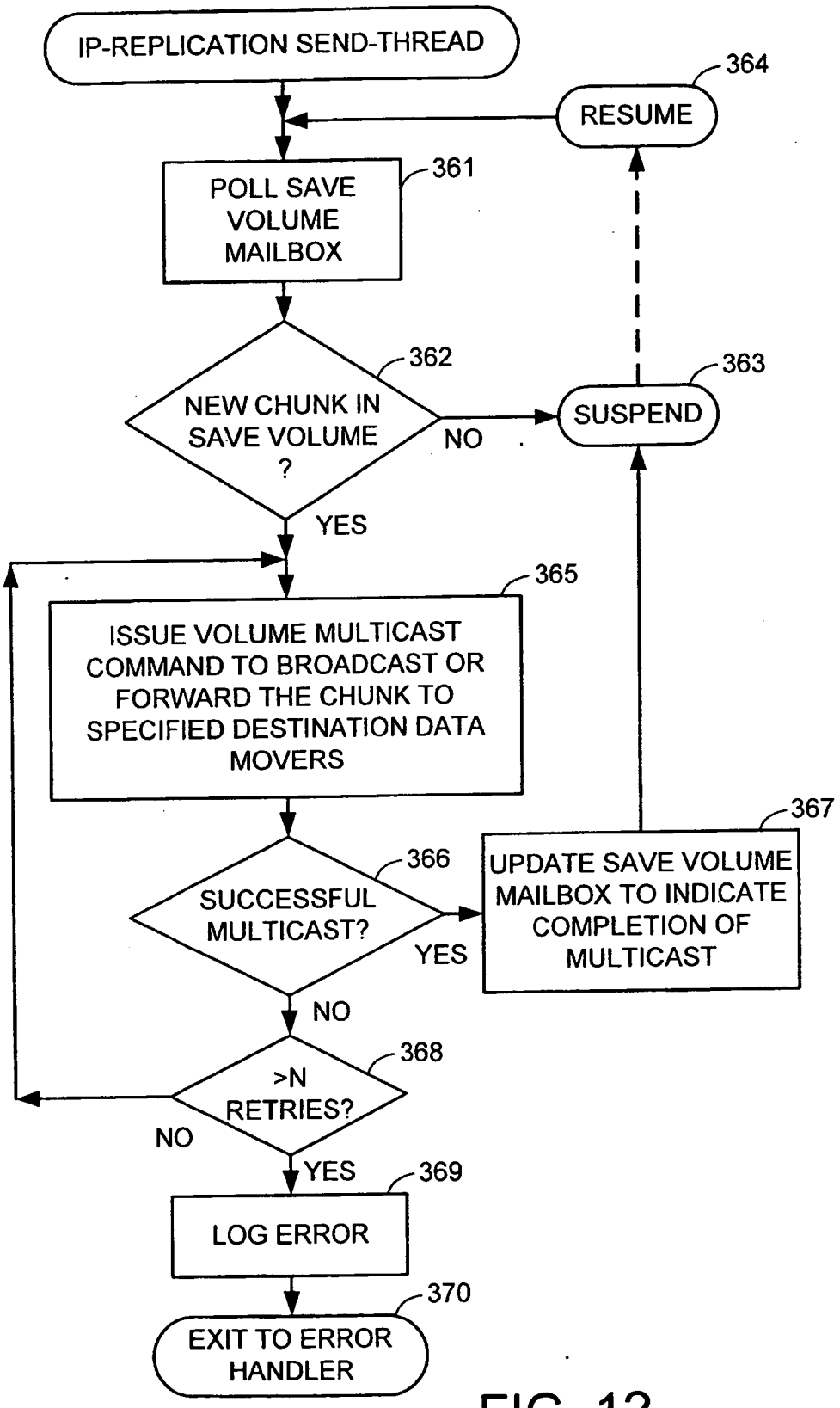

IP-REPLICATION SEND-THREAD
364
RESUME
POLL SAVE VOLUME MAILBOX
361
362
NEW CHUNK IN SAVE VOLUME ?
NO
363
SUSPEND
YES
365
ISSUE VOLUME MULTICAST COMMAND TO BROADCAST OR FORWARD THE CHUNK TO SPECIFIED DESTINATION DATA MOVERS
367
366
SUCCESSFUL MULTICAST?
YES
UPDATE SAVE VOLUME MAILBOX TO INDICATE COMPLETION OF MULTICAST
NO
368
>N RETRIES?
NO
YES
369
LOG ERROR
370
EXIT TO ERROR HANDLER

FIG. 12

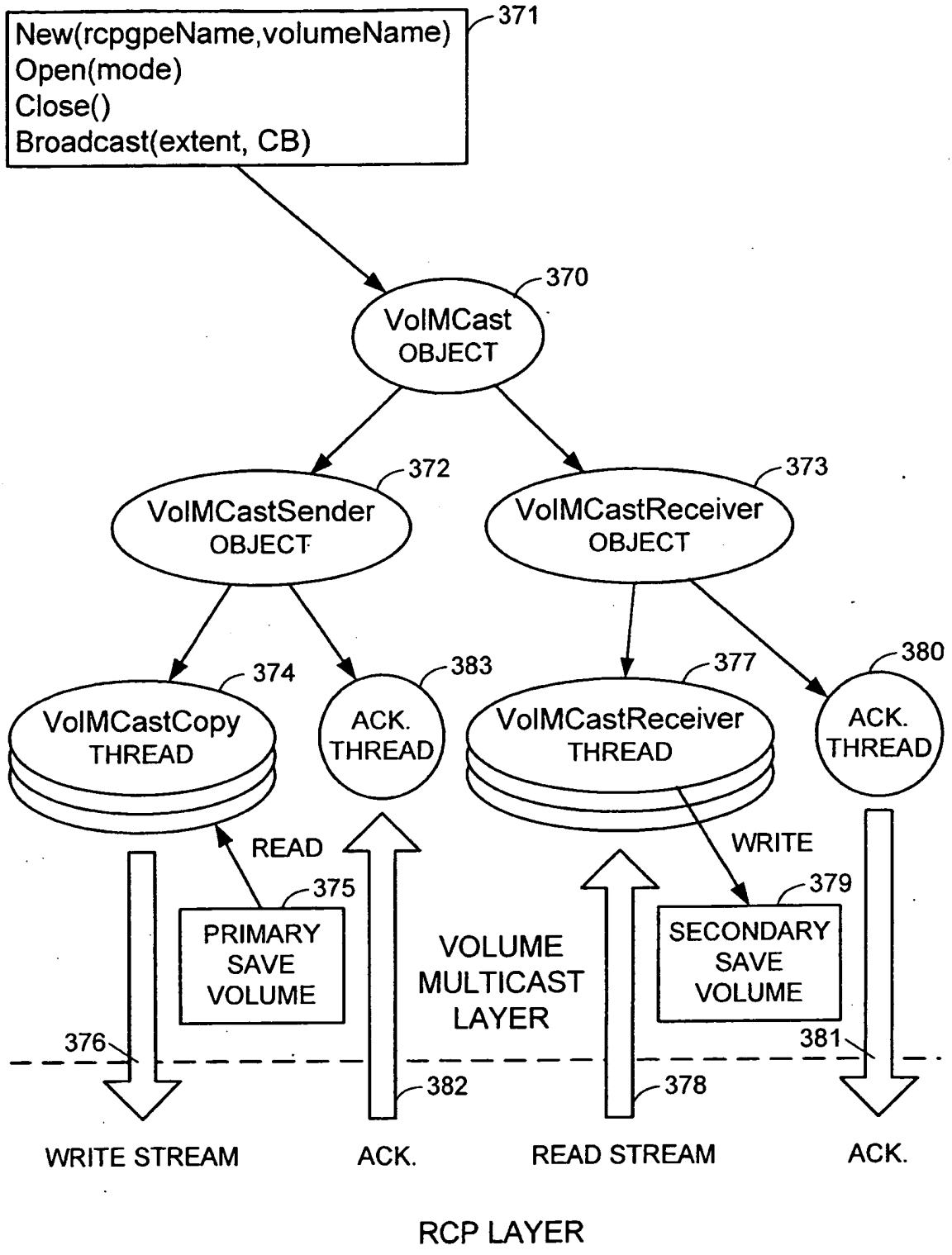

New(rcpgpeName,volumeName)
Open(mode)
Close()
Broadcast(extent, CB)
371
VolMCast OBJECT
370
VolMCastSender OBJECT
372
VolMCastReceiver OBJECT
373
VolMCastCopy THREAD
374
ACK. THREAD
383
VolMCastReceiver THREAD
377
ACK. THREAD
380
READ
WRITE
PRIMARY SAVE VOLUME
375
SECONDARY SAVE VOLUME
379
VOLUME MULTICAST LAYER
376
381
382
378
WRITE STREAM
ACK.
READ STREAM
ACK.
RCP LAYER

FIG. 13

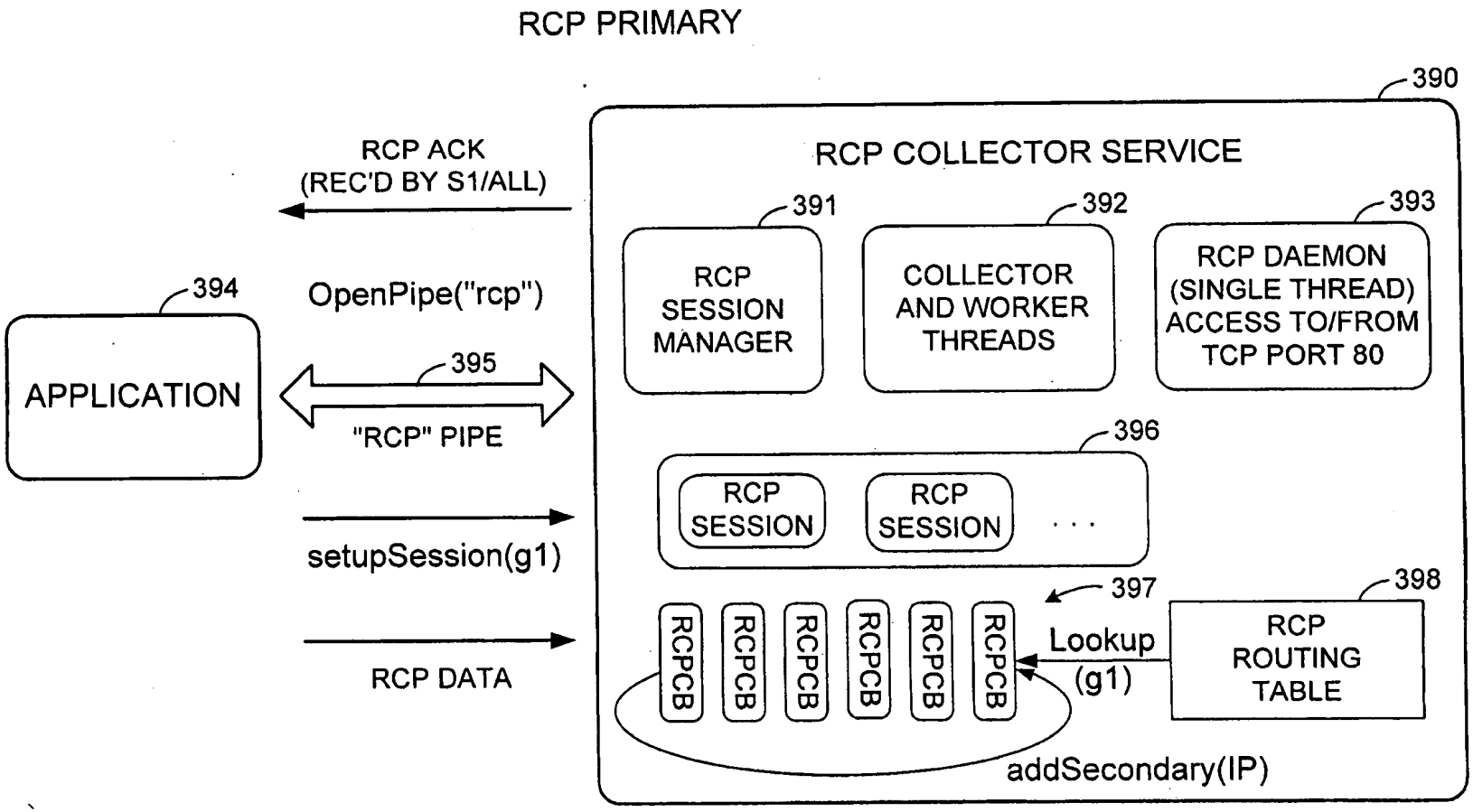

RCP PRIMARY
390
RCP COLLECTOR SERVICE
RCP ACK
(REC'D BY S1/ALL)
391
RCP
SESSION
MANAGER
392
COLLECTOR
AND WORKER
THREADS
393
RCP DAEMON
(SINGLE THREAD)
ACCESS TO/FROM
TCP PORT 80
394
APPLICATION
OpenPipe("rcp")
395
"RCP" PIPE
396
RCP
SESSION
RCP
SESSION
setupSession(g1)
397
398
RCPCB
RCPCB
RCPCB
RCPCB
RCPCB
RCPCB
Lookup
(g1)
RCP
ROUTING
TABLE
RCP DATA
addSecondary(IP)

FIG. 14

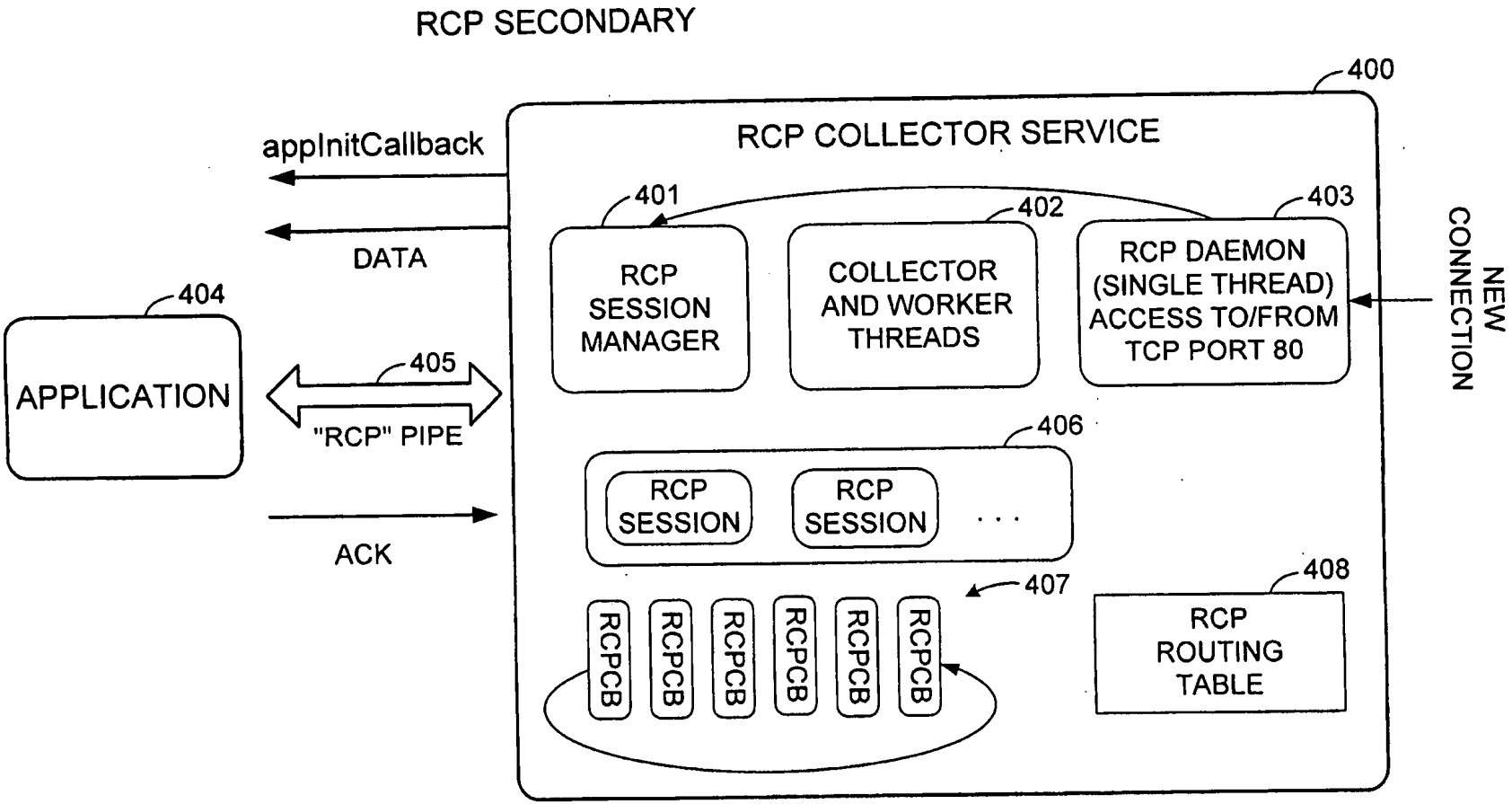

RCP SECONDARY
400
RCP COLLECTOR SERVICE
appInitCallback
DATA
401
402
403
RCP SESSION MANAGER
COLLECTOR AND WORKER THREADS
RCP DAEMON (SINGLE THREAD) ACCESS TO/FROM TCP PORT 80
NEW CONNECTION
404
APPLICATION
405
"RCP" PIPE
406
RCP SESSION
RCP SESSION
ACK
407
408
RCPCB
RCPCB
RCPCB
RCPCB
RCPCB
RCPCB
RCP ROUTING TABLE

FIG. 15

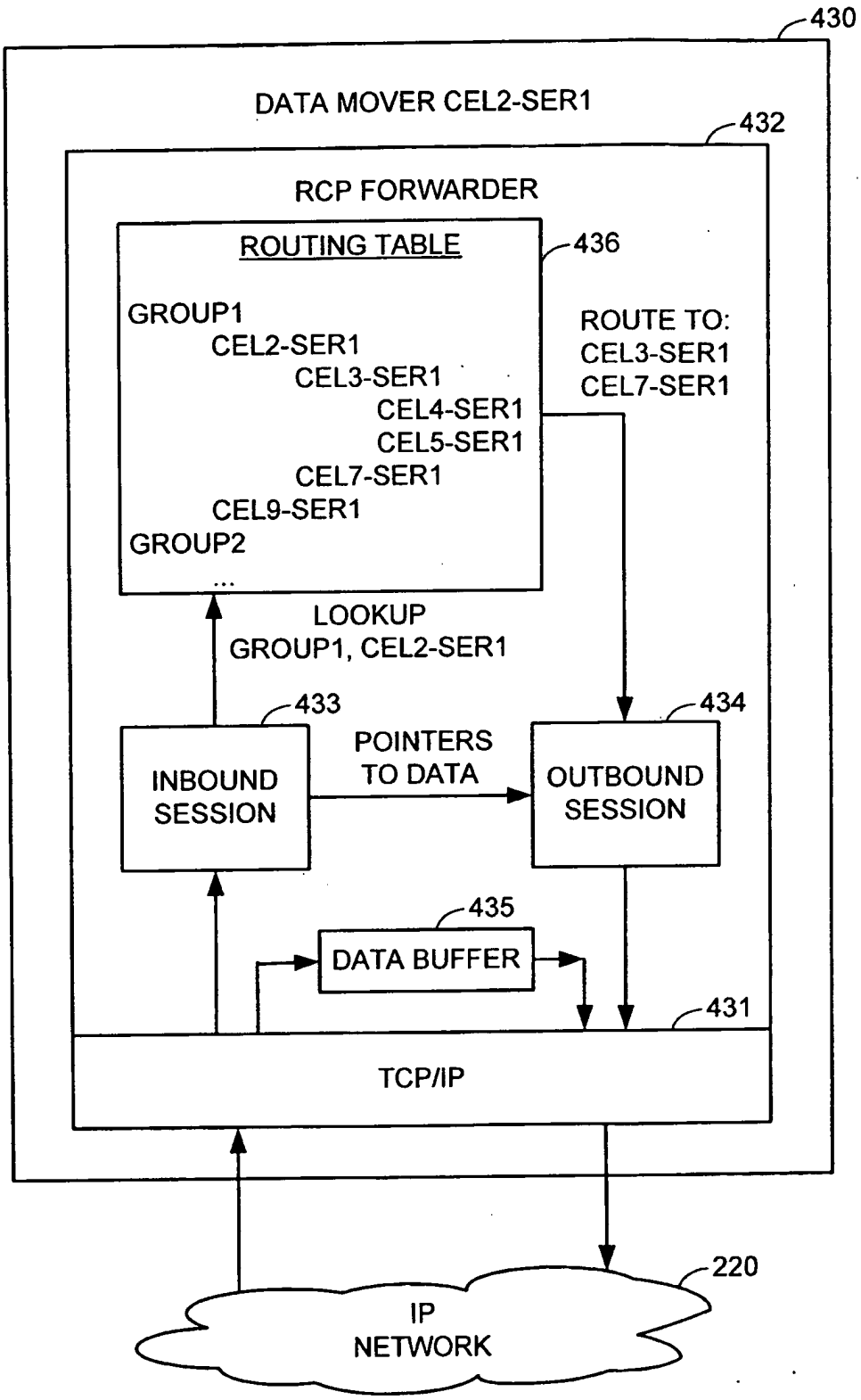

430
DATA MOVER CEL2-SER1
432
RCP FORWARDER
ROUTING TABLE
436
GROUP1
CEL2-SER1
CEL3-SER1
CEL4-SER1
CEL5-SER1
CEL7-SER1
CEL9-SER1
GROUP2
...
ROUTE TO:
CEL3-SER1
CEL7-SER1
LOOKUP
GROUP1, CEL2-SER1
433
434
INBOUND SESSION
POINTERS TO DATA
OUTBOUND SESSION
435
DATA BUFFER
431
TCP/IP
220
IP NETWORK

FIG. 16

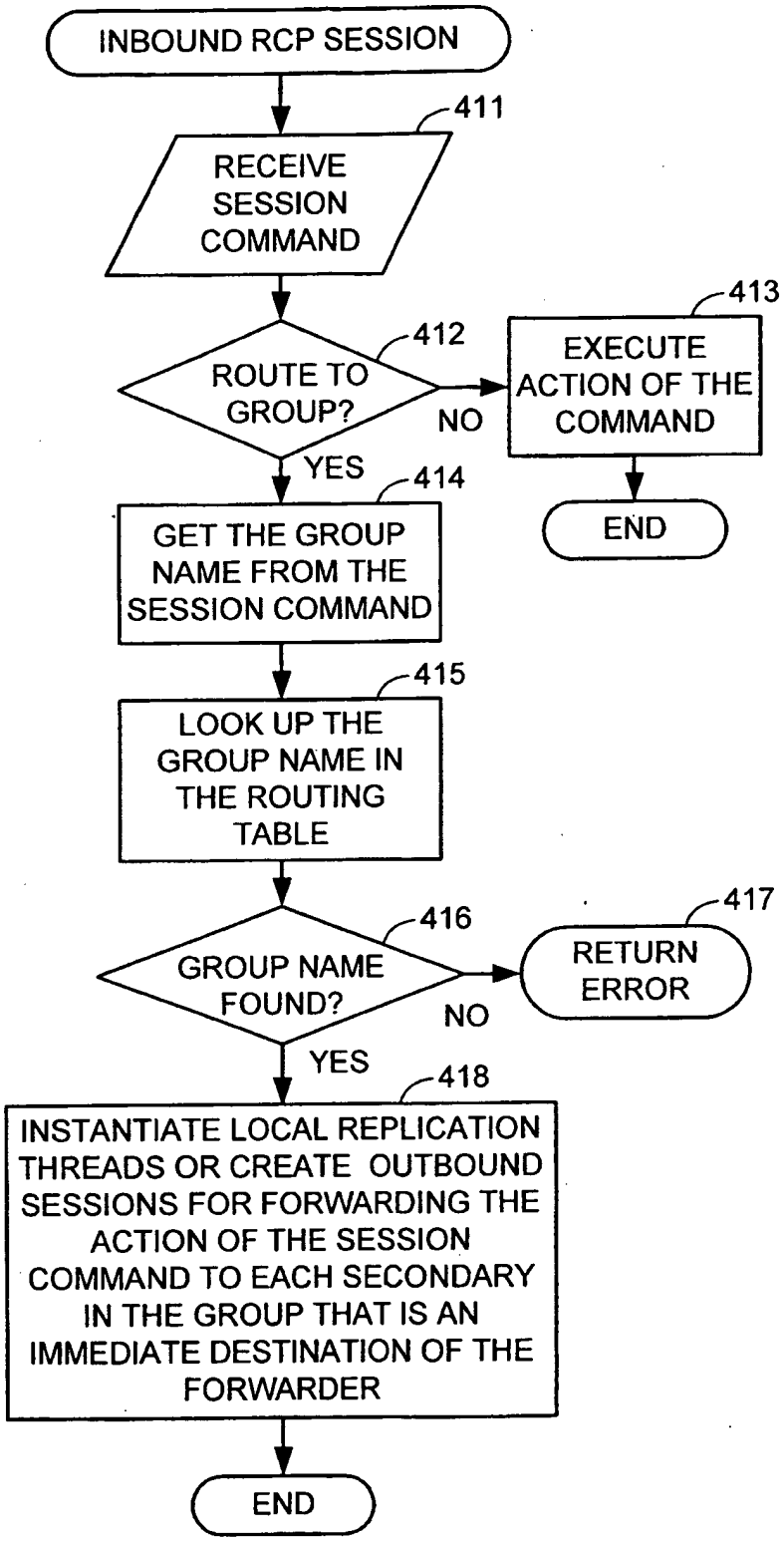

INBOUND RCP SESSION
411
RECEIVE SESSION COMMAND
412
ROUTE TO GROUP?
NO
413
EXECUTE ACTION OF THE COMMAND
END
YES
414
GET THE GROUP NAME FROM THE SESSION COMMAND
415
LOOK UP THE GROUP NAME IN THE ROUTING TABLE
416
GROUP NAME FOUND?
NO
417
RETURN ERROR
YES
418
INSTANTIATE LOCAL REPLICATION THREADS OR CREATE OUTBOUND SESSIONS FOR FORWARDING THE ACTION OF THE SESSION COMMAND TO EACH SECONDARY IN THE GROUP THAT IS AN IMMEDIATE DESTINATION OF THE FORWARDER
END

FIG. 17

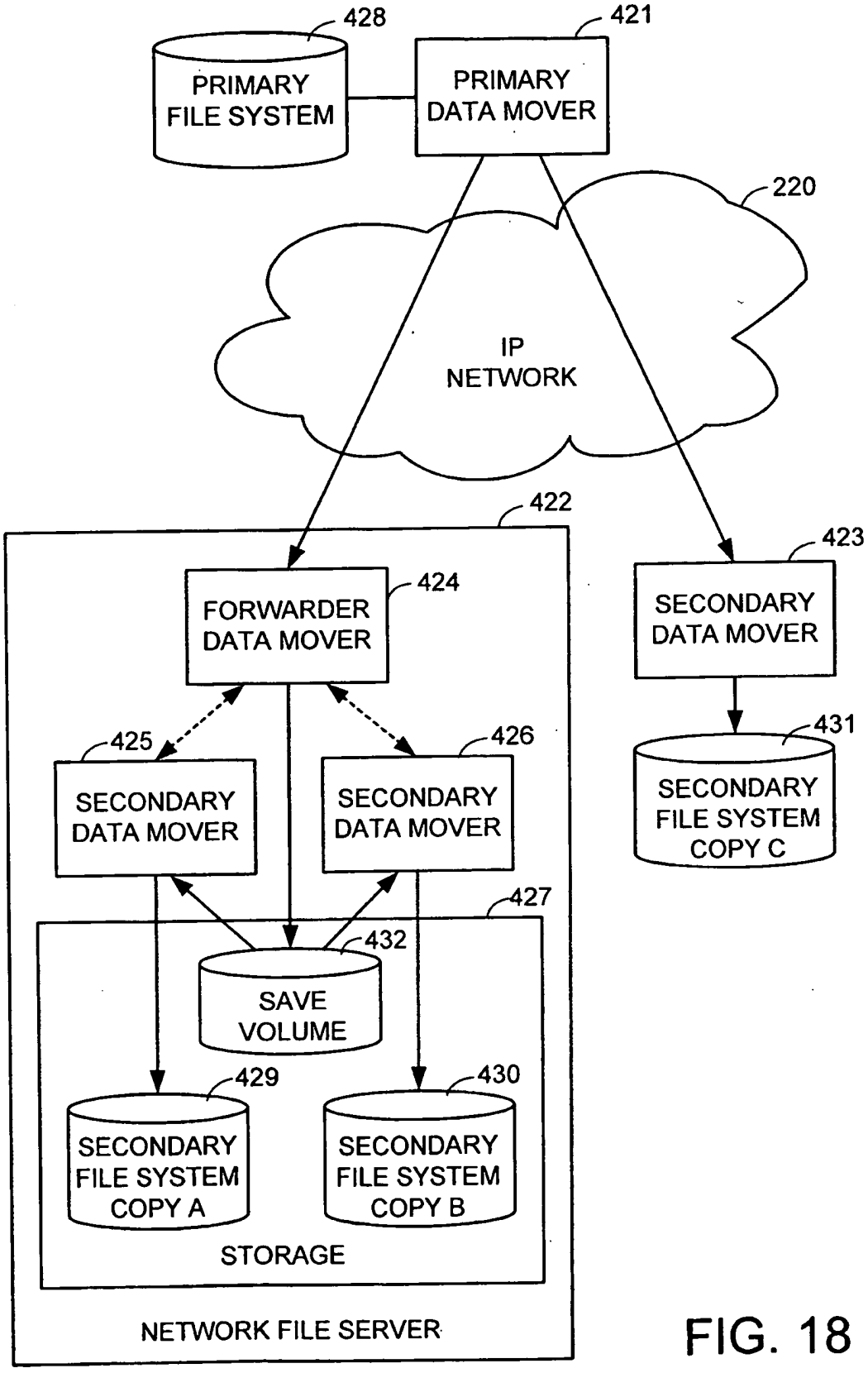

428
PRIMARY FILE SYSTEM
421
PRIMARY DATA MOVER
220
IP NETWORK
422
423
FORWARDER DATA MOVER
424
SECONDARY DATA MOVER
425
SECONDARY DATA MOVER
426
SECONDARY DATA MOVER
431
SECONDARY FILE SYSTEM COPY C
427
432
SAVE VOLUME
429
SECONDARY FILE SYSTEM COPY A
430
SECONDARY FILE SYSTEM COPY B
STORAGE
NETWORK FILE SERVER

FIG. 18

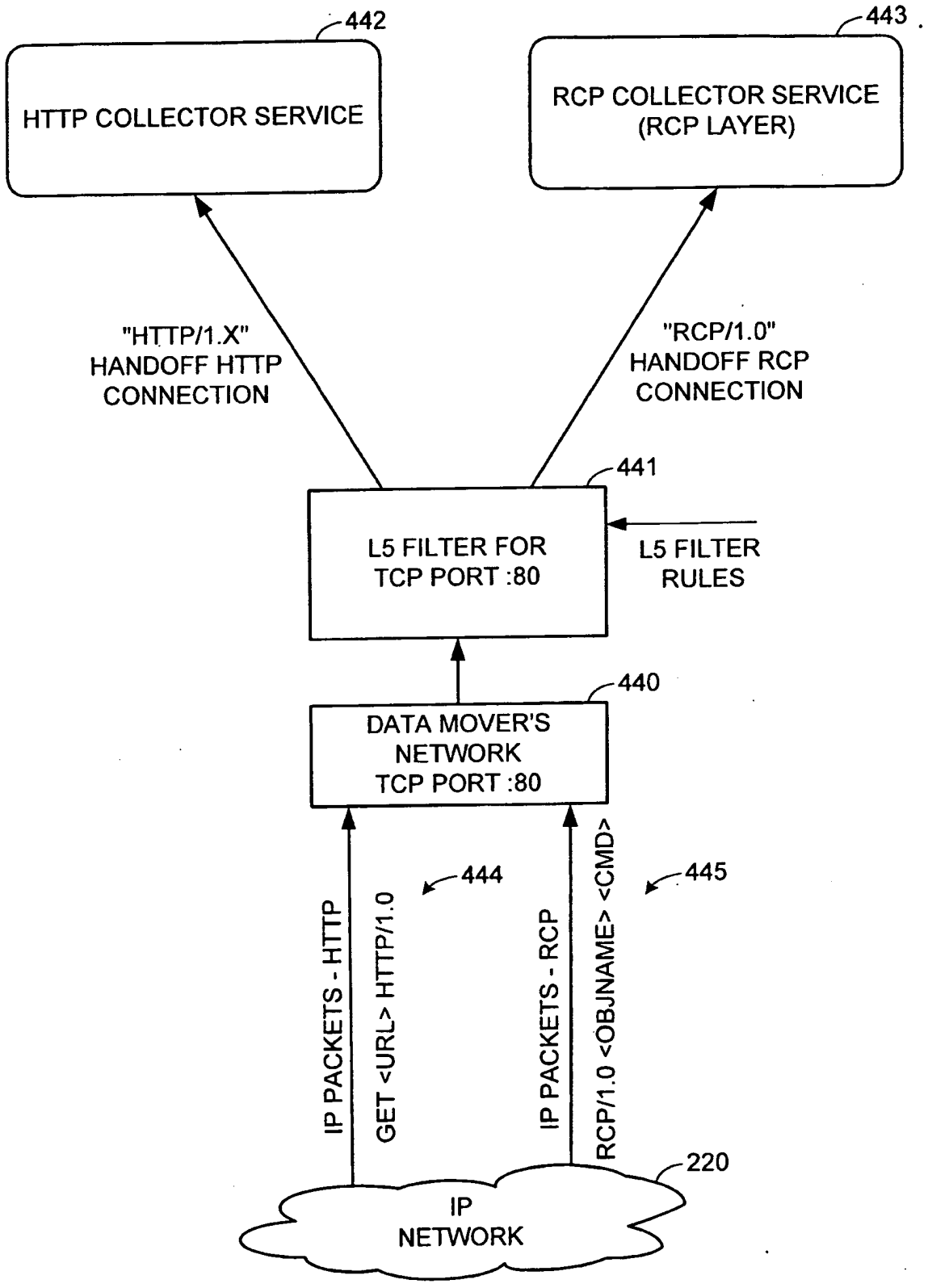

442
HTTP COLLECTOR SERVICE
443
RCP COLLECTOR SERVICE
(RCP LAYER)
"HTTP/1.X"
HANDOFF HTTP
CONNECTION
"RCP/1.0"
HANDOFF RCP
CONNECTION
441
L5 FILTER FOR
TCP PORT :80
L5 FILTER
RULES
440
DATA MOVER'S
NETWORK
TCP PORT :80
444
445
IP PACKETS - HTTP
GET <URL> HTTP/1.0
IP PACKETS - RCP
RCP/1.0 <OBJNAME> <CMD>
220
IP
NETWORK

FIG. 19

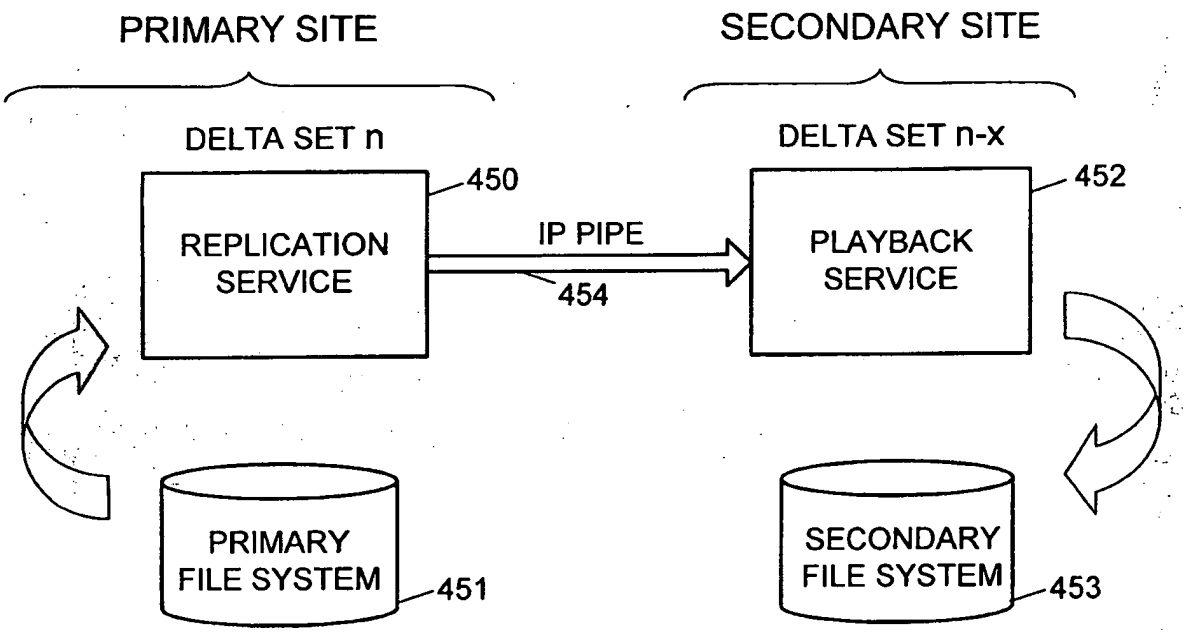

PRIMARY SITE
SECONDARY SITE
DELTA SET n
DELTA SET n-x
450
452
REPLICATION SERVICE
IP PIPE
454
PLAYBACK SERVICE
PRIMARY FILE SYSTEM
451
SECONDARY FILE SYSTEM
453

FIG. 20

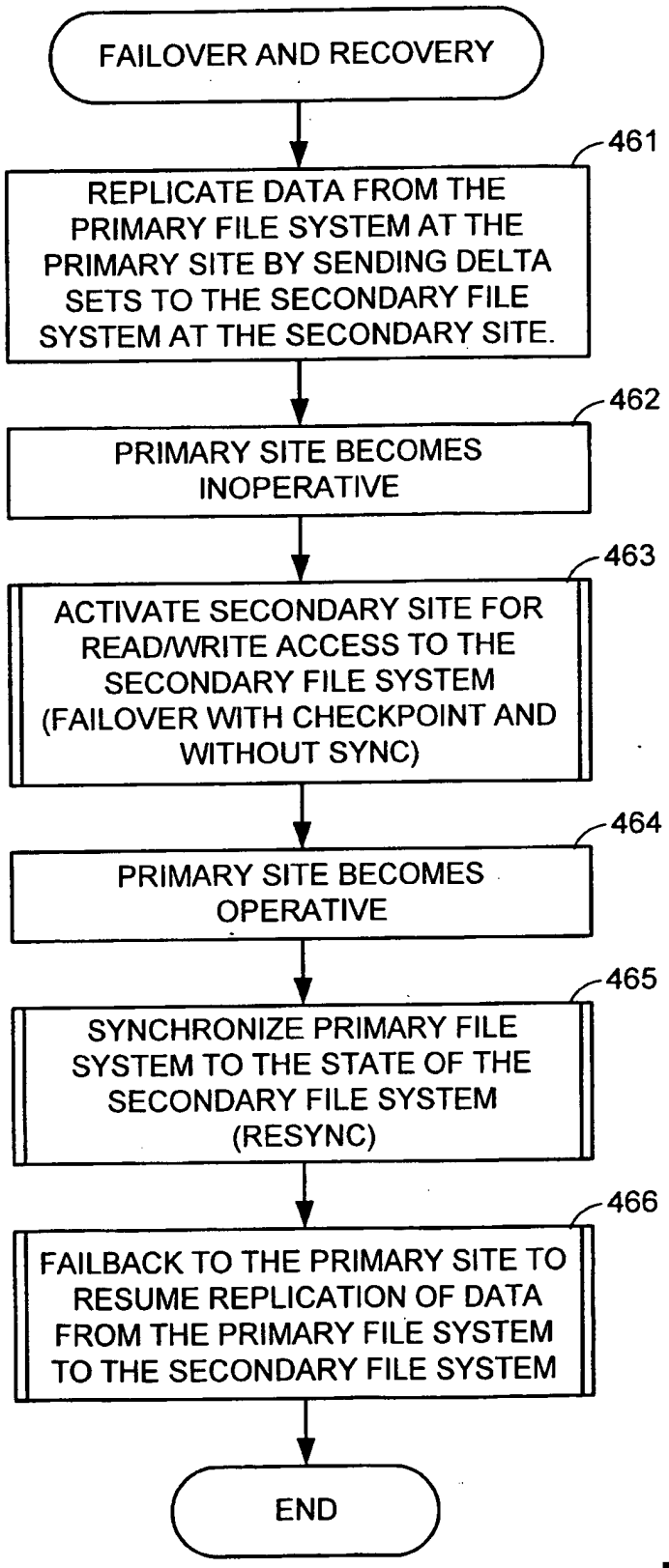

FAILOVER AND RECOVERY
461
REPLICATE DATA FROM THE PRIMARY FILE SYSTEM AT THE PRIMARY SITE BY SENDING DELTA SETS TO THE SECONDARY FILE SYSTEM AT THE SECONDARY SITE.
462
PRIMARY SITE BECOMES INOPERATIVE
463
ACTIVATE SECONDARY SITE FOR READ/WRITE ACCESS TO THE SECONDARY FILE SYSTEM (FAILOVER WITH CHECKPOINT AND WITHOUT SYNC)
464
PRIMARY SITE BECOMES OPERATIVE
465
SYNCHRONIZE PRIMARY FILE SYSTEM TO THE STATE OF THE SECONDARY FILE SYSTEM (RESYNC)
466
FAILBACK TO THE PRIMARY SITE TO RESUME REPLICATION OF DATA FROM THE PRIMARY FILE SYSTEM TO THE SECONDARY FILE SYSTEM
END

FIG. 21

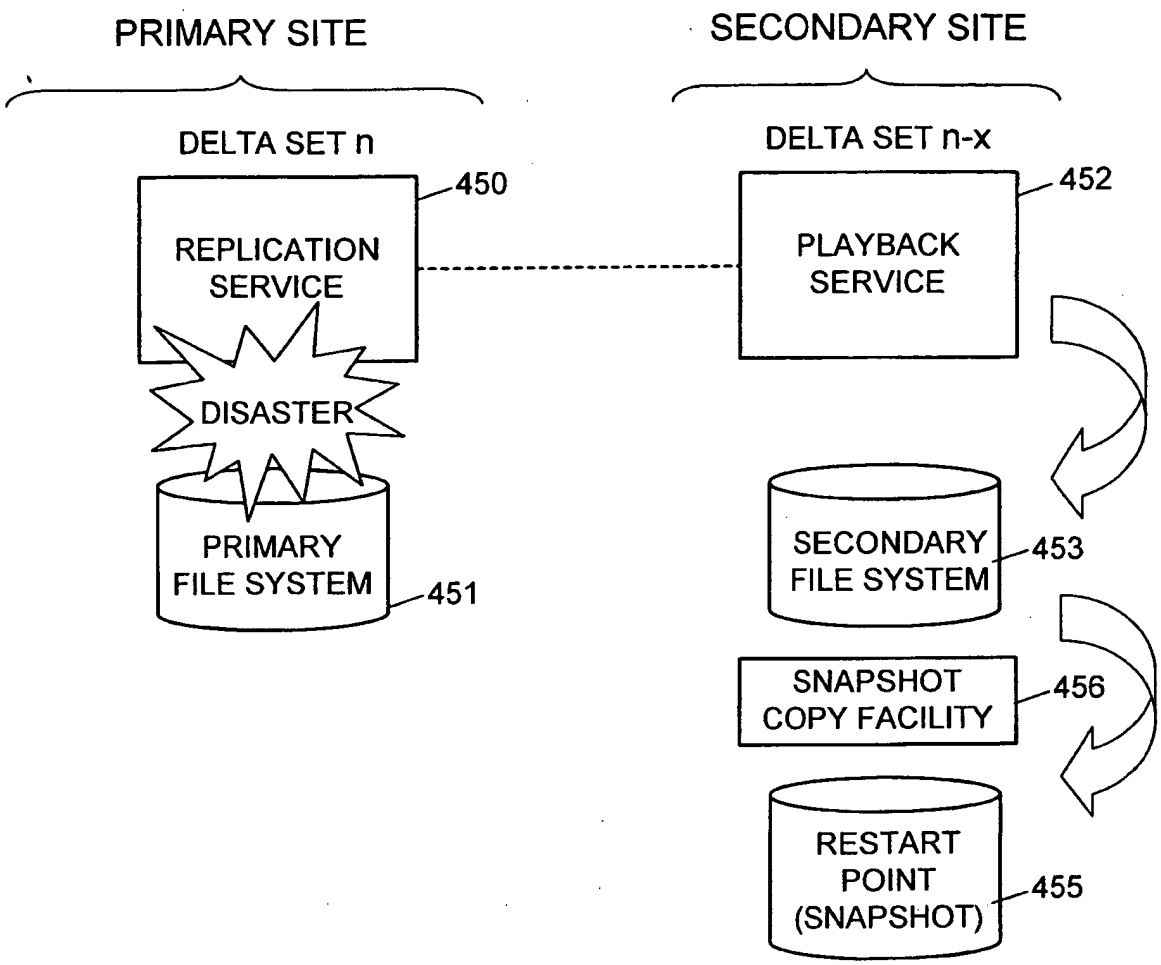


FIG. 22

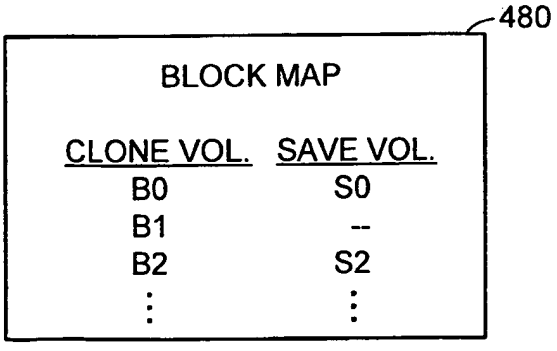


FIG. 24
(PRIOR ART)

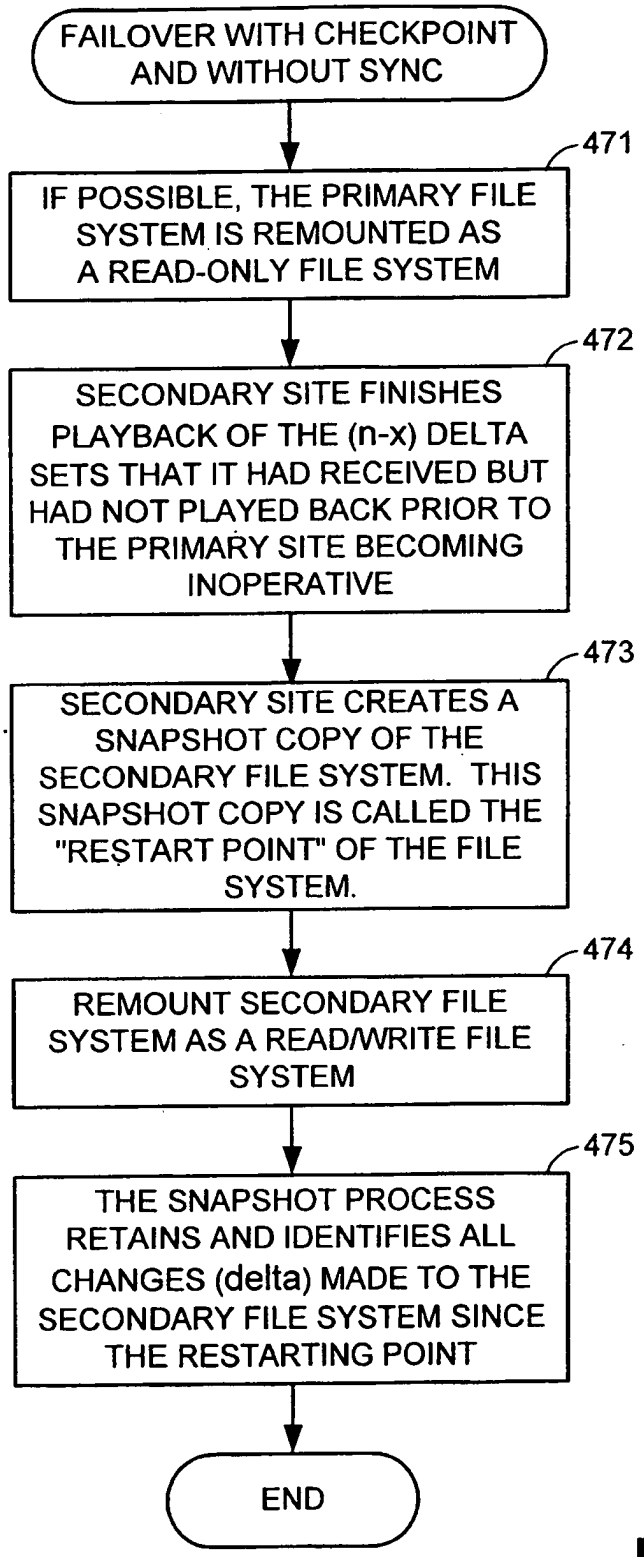

FAILOVER WITH CHECKPOINT AND WITHOUT SYNC
471
IF POSSIBLE, THE PRIMARY FILE SYSTEM IS REMOUNTED AS A READ-ONLY FILE SYSTEM
472
SECONDARY SITE FINISHES PLAYBACK OF THE (n-x) DELTA SETS THAT IT HAD RECEIVED BUT HAD NOT PLAYED BACK PRIOR TO THE PRIMARY SITE BECOMING INOPERATIVE
473
SECONDARY SITE CREATES A SNAPSHOT COPY OF THE SECONDARY FILE SYSTEM. THIS SNAPSHOT COPY IS CALLED THE "RESTART POINT" OF THE FILE SYSTEM.
474
REMOUNT SECONDARY FILE SYSTEM AS A READ/WRITE FILE SYSTEM
475
THE SNAPSHOT PROCESS RETAINS AND IDENTIFIES ALL CHANGES (delta) MADE TO THE SECONDARY FILE SYSTEM SINCE THE RESTARTING POINT
END

FIG. 23

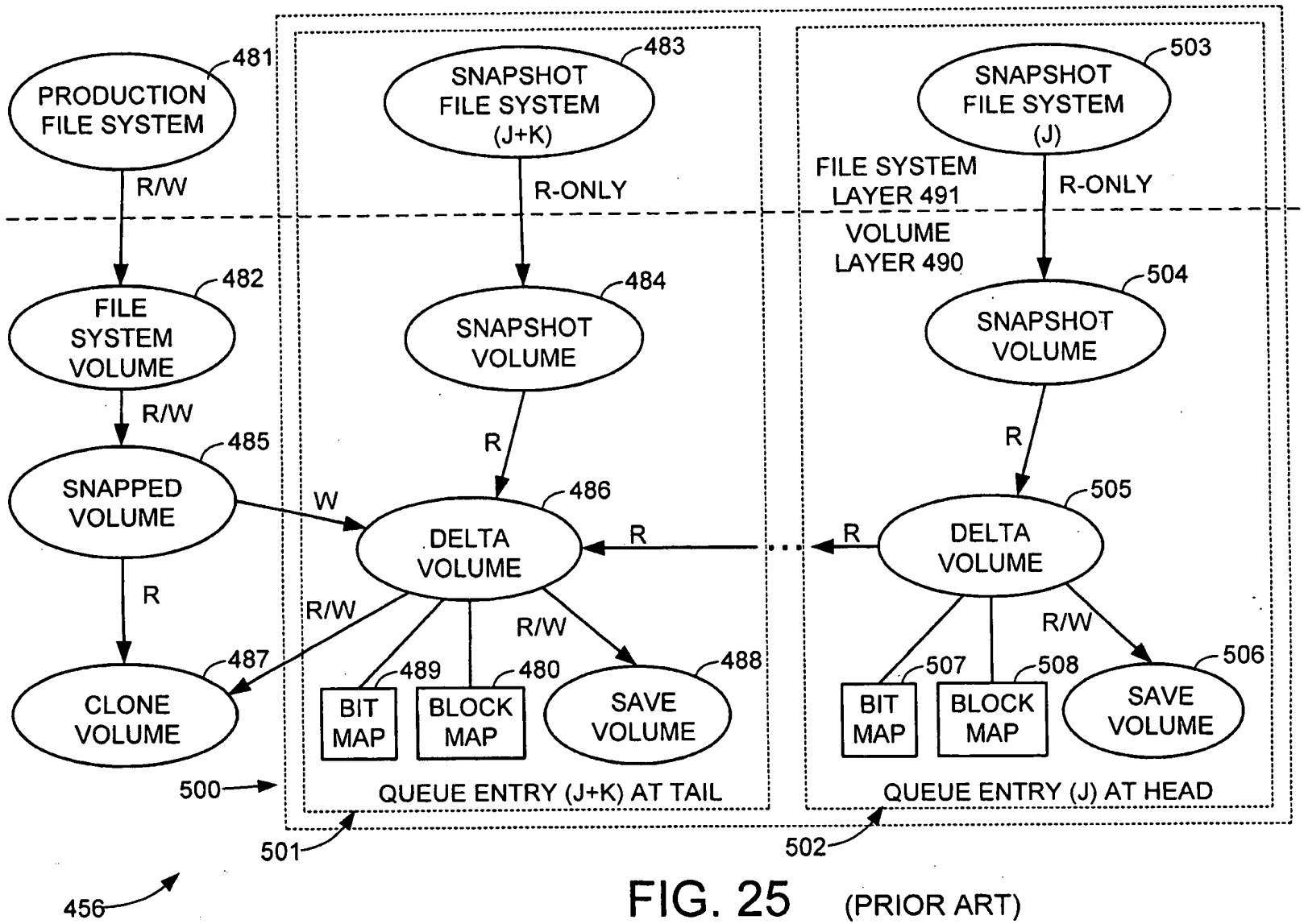


FIG. 25 (PRIOR ART)

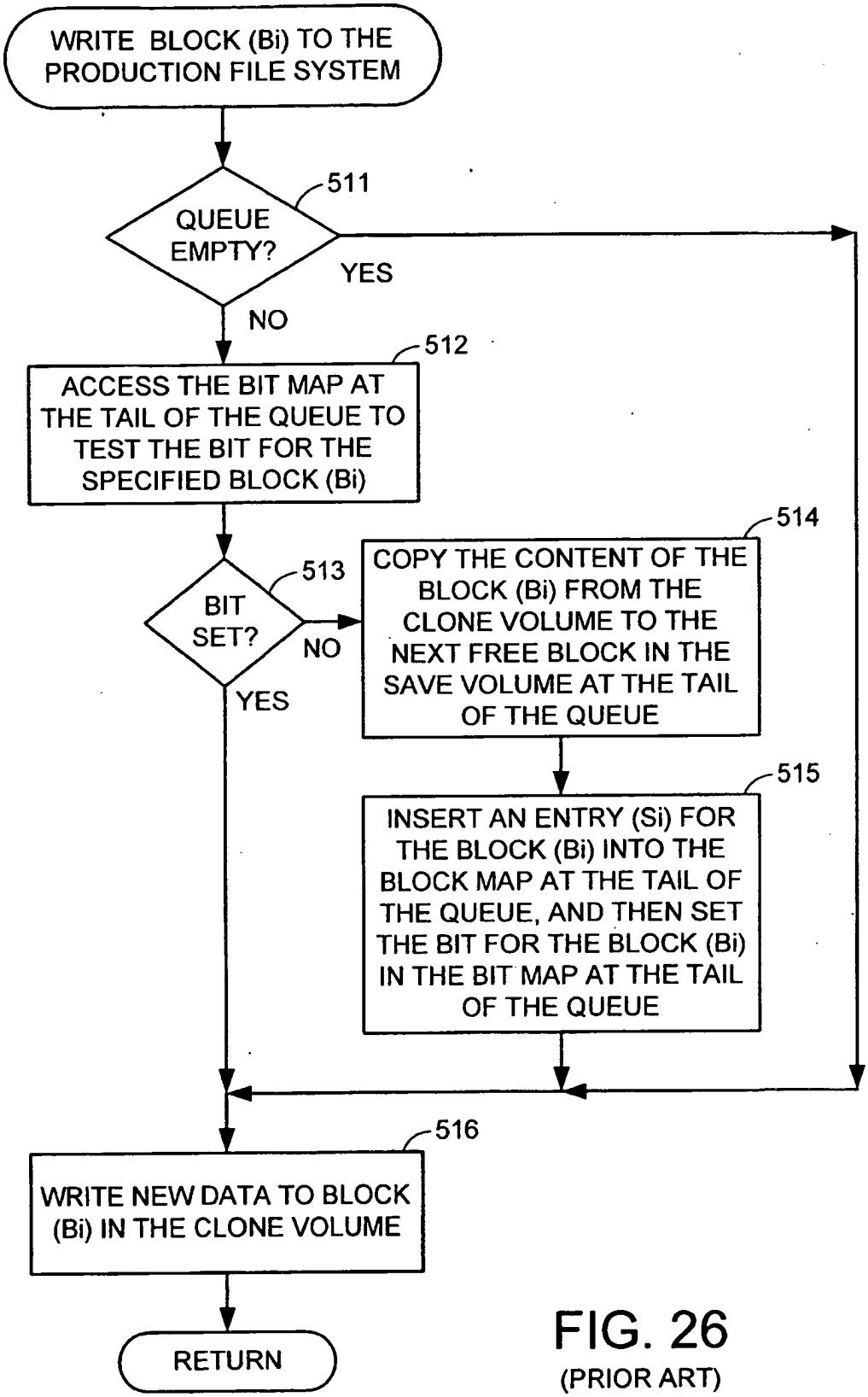


FIG. 26
(PRIOR ART)

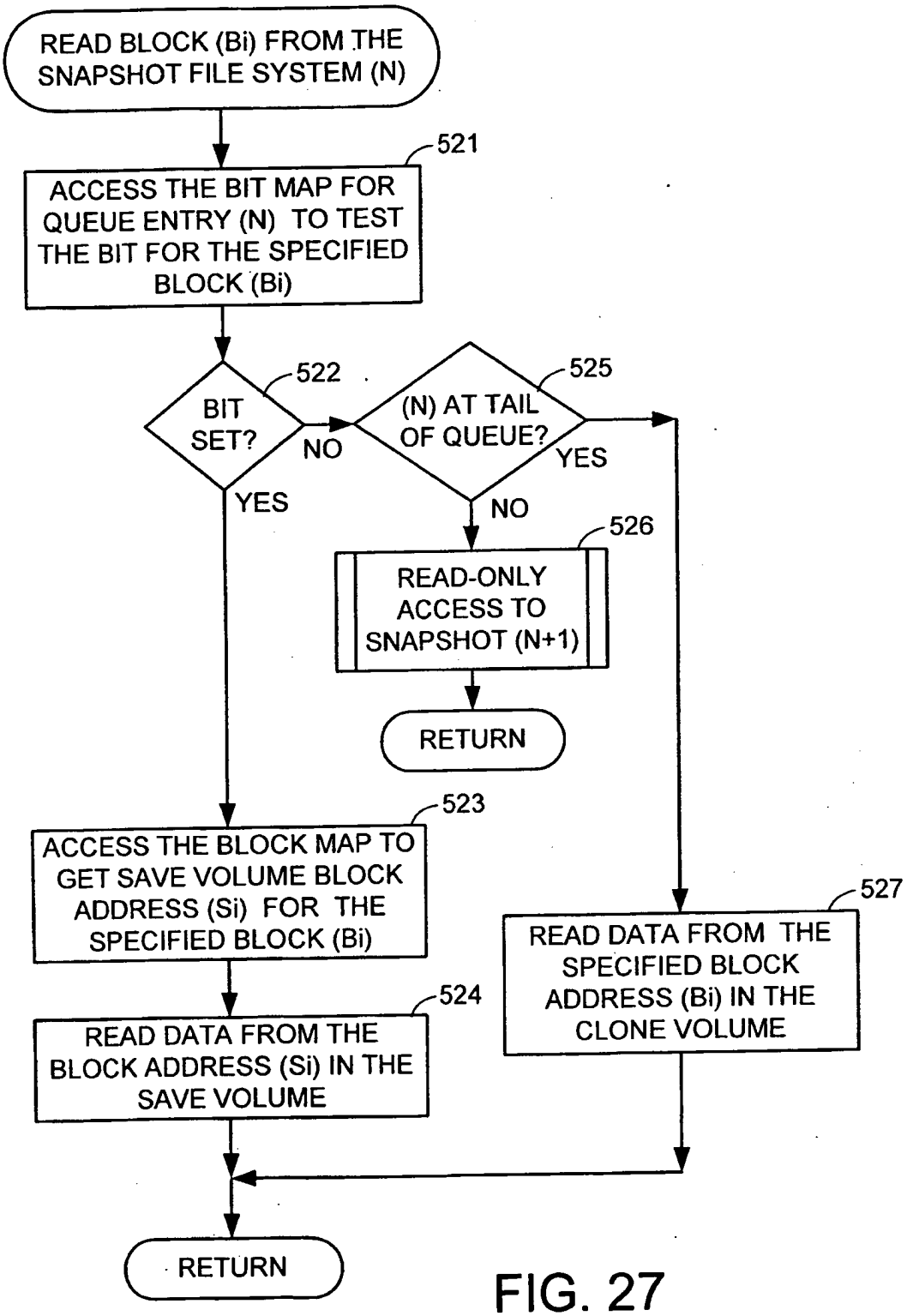


FIG. 27
(PRIOR ART)

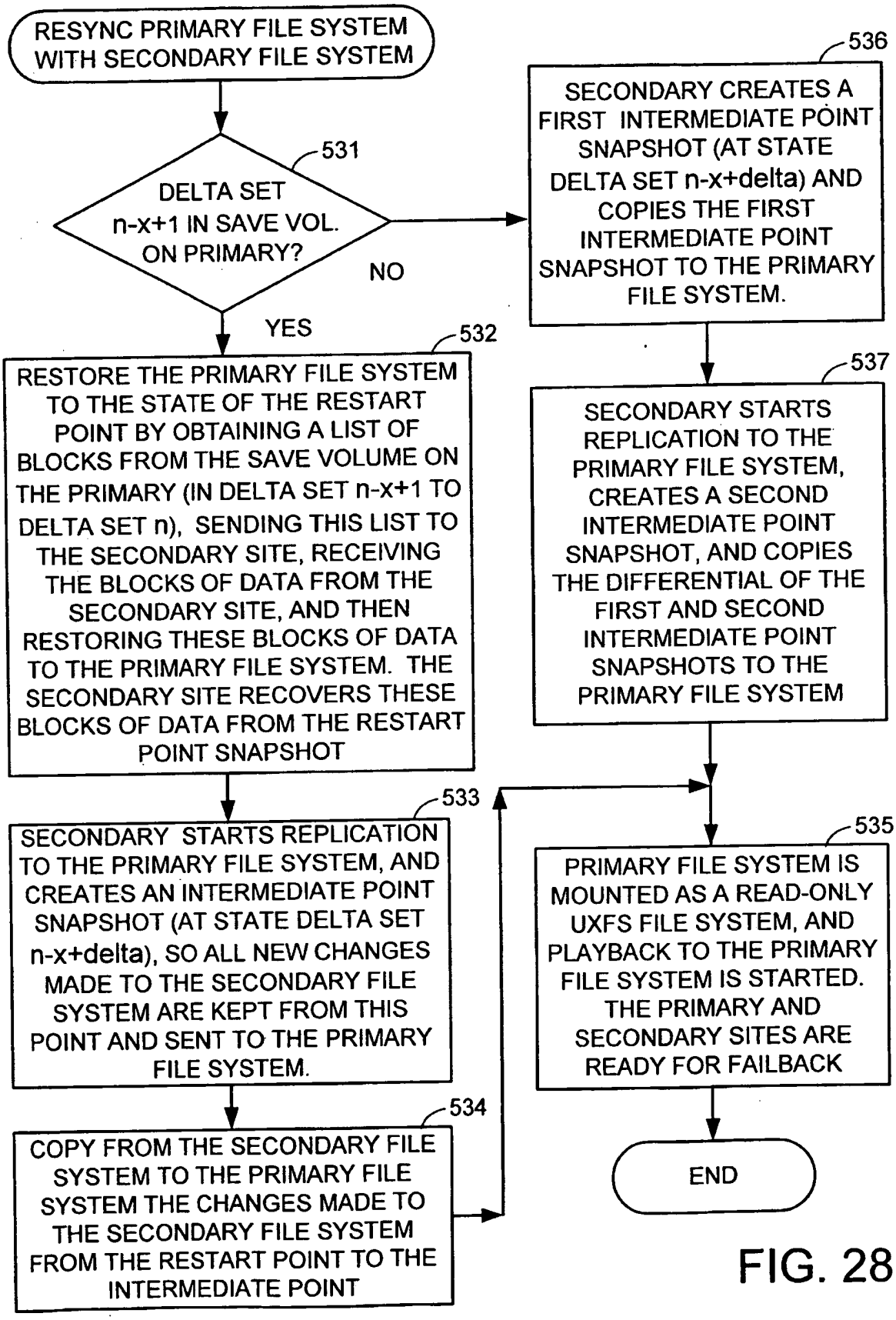

RESYNC PRIMARY FILE SYSTEM WITH SECONDARY FILE SYSTEM
531
DELTA SET n-x+1 IN SAVE VOL. ON PRIMARY?
NO
YES
536
SECONDARY CREATES A FIRST INTERMEDIATE POINT SNAPSHOT (AT STATE DELTA SET n-x+delta) AND COPIES THE FIRST INTERMEDIATE POINT SNAPSHOT TO THE PRIMARY FILE SYSTEM.
532
RESTORE THE PRIMARY FILE SYSTEM TO THE STATE OF THE RESTART POINT BY OBTAINING A LIST OF BLOCKS FROM THE SAVE VOLUME ON THE PRIMARY (IN DELTA SET n-x+1 TO DELTA SET n), SENDING THIS LIST TO THE SECONDARY SITE, RECEIVING THE BLOCKS OF DATA FROM THE SECONDARY SITE, AND THEN RESTORING THESE BLOCKS OF DATA TO THE PRIMARY FILE SYSTEM. THE SECONDARY SITE RECOVERS THESE BLOCKS OF DATA FROM THE RESTART POINT SNAPSHOT
537
SECONDARY STARTS REPLICATION TO THE PRIMARY FILE SYSTEM, CREATES A SECOND INTERMEDIATE POINT SNAPSHOT, AND COPIES THE DIFFERENTIAL OF THE FIRST AND SECOND INTERMEDIATE POINT SNAPSHOTS TO THE PRIMARY FILE SYSTEM
533
SECONDARY STARTS REPLICATION TO THE PRIMARY FILE SYSTEM, AND CREATES AN INTERMEDIATE POINT SNAPSHOT (AT STATE DELTA SET n-x+delta), SO ALL NEW CHANGES MADE TO THE SECONDARY FILE SYSTEM ARE KEPT FROM THIS POINT AND SENT TO THE PRIMARY FILE SYSTEM.
535
PRIMARY FILE SYSTEM IS MOUNTED AS A READ-ONLY UXFS FILE SYSTEM, AND PLAYBACK TO THE PRIMARY FILE SYSTEM IS STARTED. THE PRIMARY AND SECONDARY SITES ARE READY FOR FAILBACK
534
COPY FROM THE SECONDARY FILE SYSTEM TO THE PRIMARY FILE SYSTEM THE CHANGES MADE TO THE SECONDARY FILE SYSTEM FROM THE RESTART POINT TO THE INTERMEDIATE POINT
END

FIG. 28

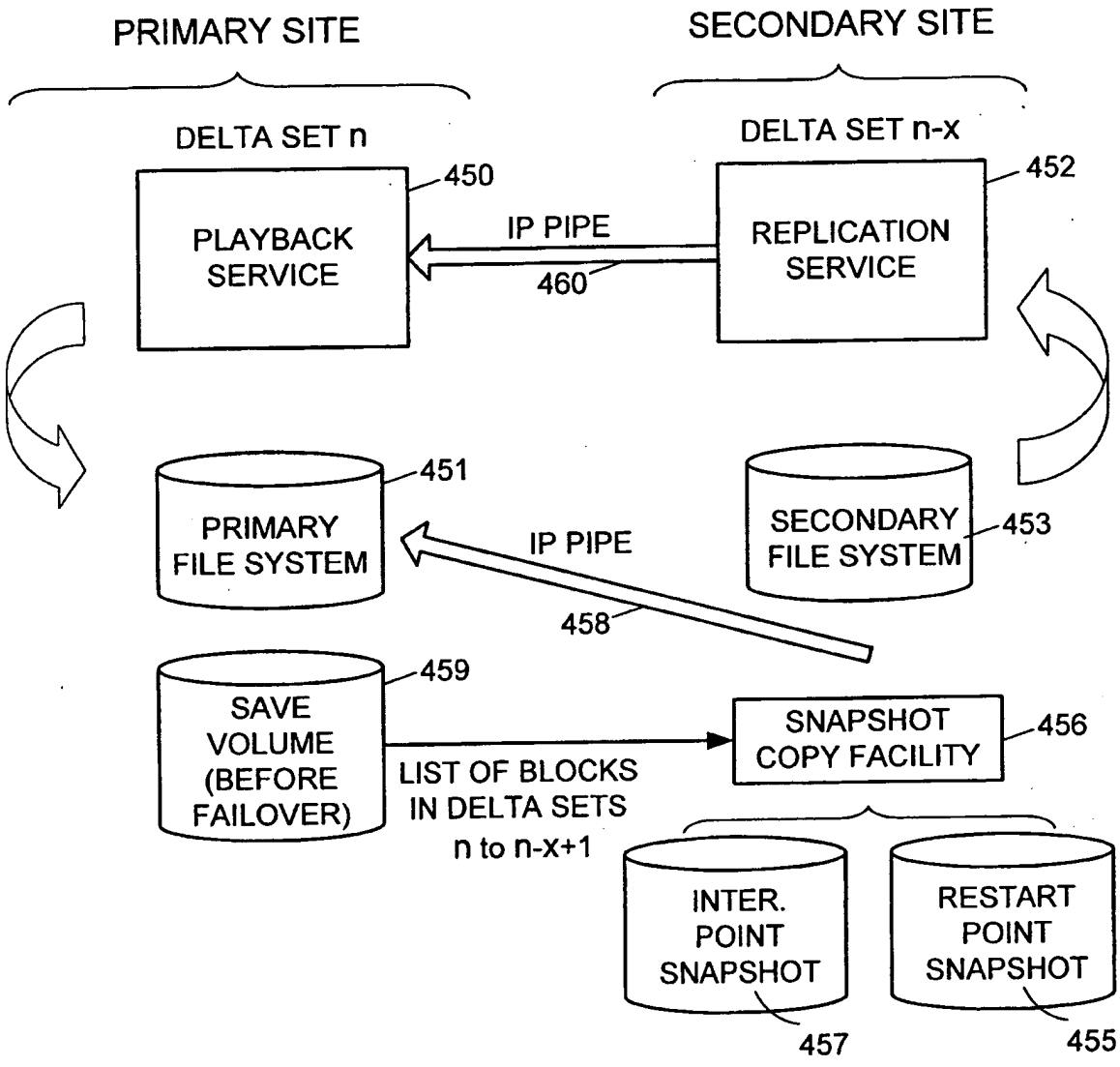

PRIMARY SITE
SECONDARY SITE
DELTA SET n
DELTA SET n-x
450
452
PLAYBACK SERVICE
IP PIPE
460
REPLICATION SERVICE
451
PRIMARY FILE SYSTEM
IP PIPE
458
SECONDARY FILE SYSTEM
453
459
SAVE VOLUME (BEFORE FAILOVER)
LIST OF BLOCKS IN DELTA SETS n to n-x+1
SNAPSHOT COPY FACILITY
456
INTER. POINT SNAPSHOT
RESTART POINT SNAPSHOT
457
455

FIG. 29

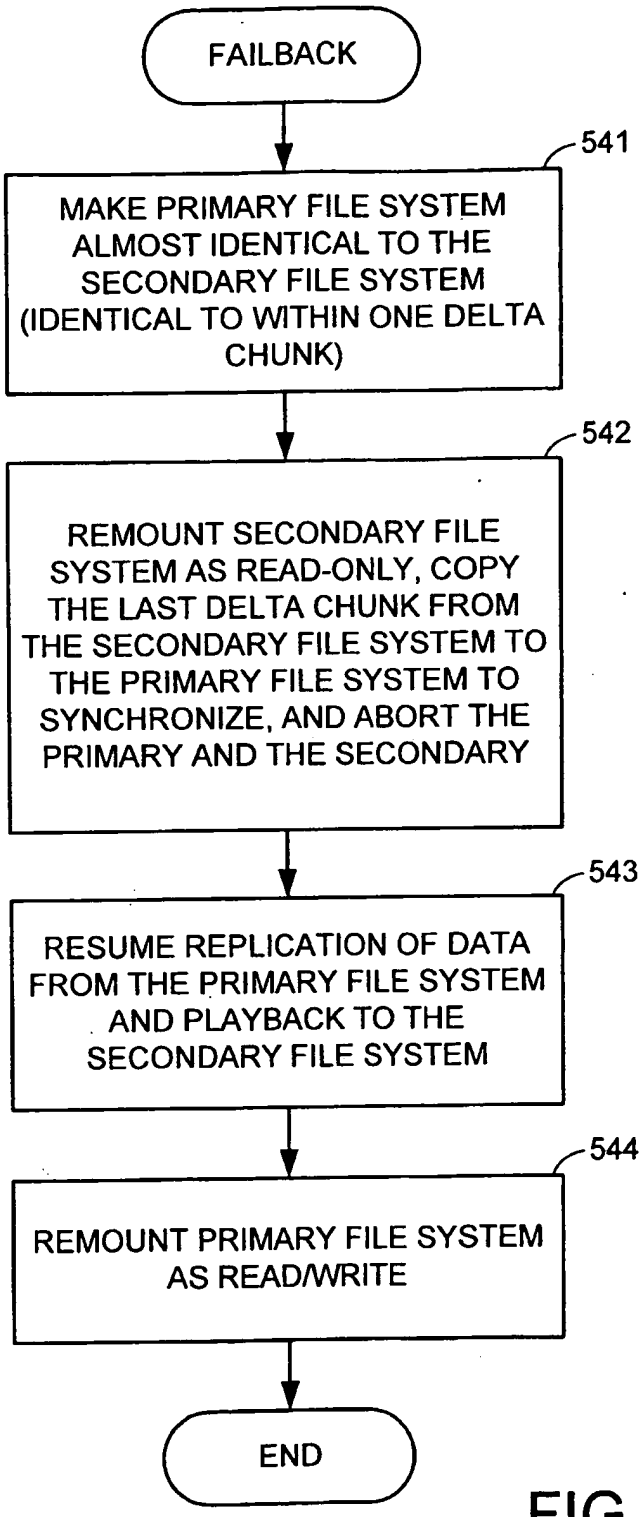

FAILBACK
541
MAKE PRIMARY FILE SYSTEM ALMOST IDENTICAL TO THE SECONDARY FILE SYSTEM (IDENTICAL TO WITHIN ONE DELTA CHUNK)
542
REMOUNT SECONDARY FILE SYSTEM AS READ-ONLY, COPY THE LAST DELTA CHUNK FROM THE SECONDARY FILE SYSTEM TO THE PRIMARY FILE SYSTEM TO SYNCHRONIZE, AND ABORT THE PRIMARY AND THE SECONDARY
543
RESUME REPLICATION OF DATA FROM THE PRIMARY FILE SYSTEM AND PLAYBACK TO THE SECONDARY FILE SYSTEM
544
REMOUNT PRIMARY FILE SYSTEM AS READ/WRITE
END

FIG. 30

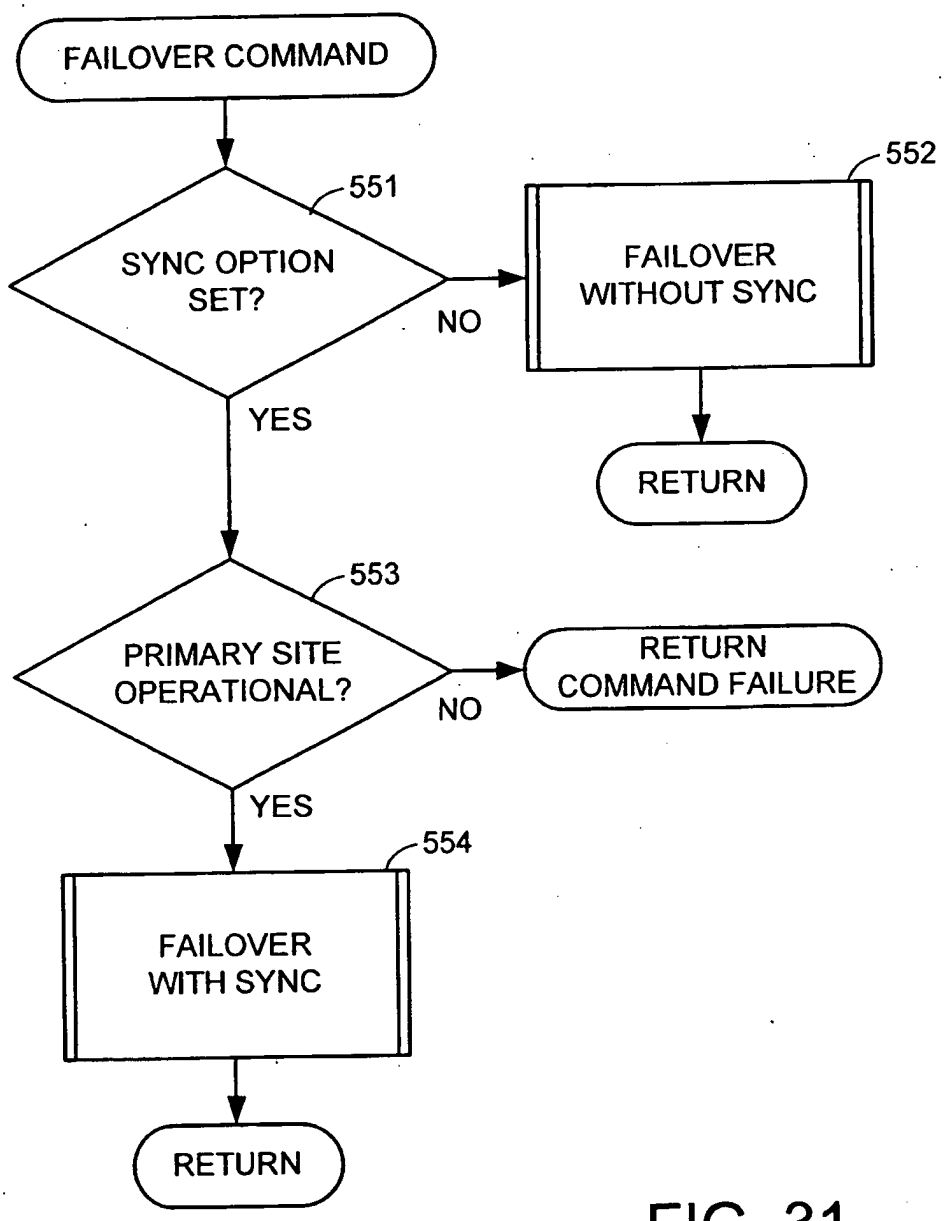

FAILOVER COMMAND
551
SYNC OPTION SET?
NO
552
FAILOVER WITHOUT SYNC
RETURN
YES
553
PRIMARY SITE OPERATIONAL?
NO
RETURN COMMAND FAILURE
YES
554
FAILOVER WITH SYNC
RETURN

FIG. 31

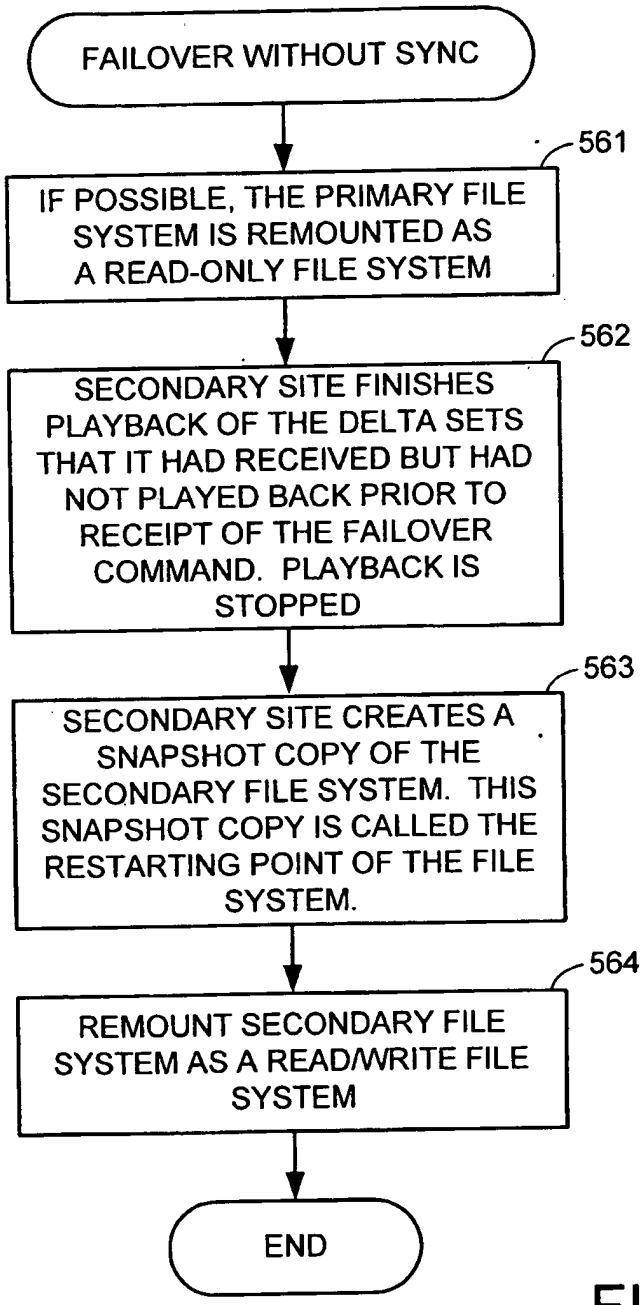

FAILOVER WITHOUT SYNC
561
IF POSSIBLE, THE PRIMARY FILE SYSTEM IS REMOUNTED AS A READ-ONLY FILE SYSTEM
562
SECONDARY SITE FINISHES PLAYBACK OF THE DELTA SETS THAT IT HAD RECEIVED BUT HAD NOT PLAYED BACK PRIOR TO RECEIPT OF THE FAILOVER COMMAND. PLAYBACK IS STOPPED
563
SECONDARY SITE CREATES A SNAPSHOT COPY OF THE SECONDARY FILE SYSTEM. THIS SNAPSHOT COPY IS CALLED THE RESTARTING POINT OF THE FILE SYSTEM.
564
REMOUNT SECONDARY FILE SYSTEM AS A READ/WRITE FILE SYSTEM
END

FIG. 32

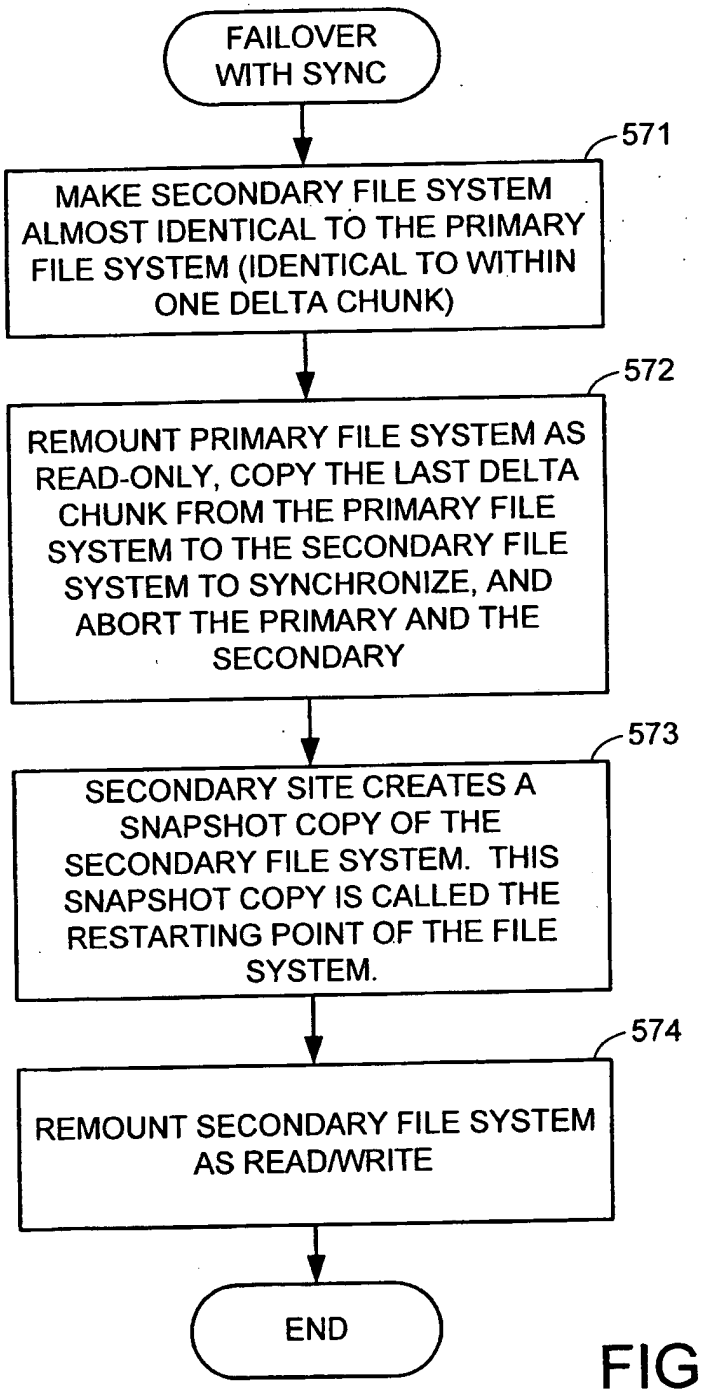

FAILOVER WITH SYNC
571
MAKE SECONDARY FILE SYSTEM ALMOST IDENTICAL TO THE PRIMARY FILE SYSTEM (IDENTICAL TO WITHIN ONE DELTA CHUNK)
572
REMOUNT PRIMARY FILE SYSTEM AS READ-ONLY, COPY THE LAST DELTA CHUNK FROM THE PRIMARY FILE SYSTEM TO THE SECONDARY FILE SYSTEM TO SYNCHRONIZE, AND ABORT THE PRIMARY AND THE SECONDARY
573
SECONDARY SITE CREATES A SNAPSHOT COPY OF THE SECONDARY FILE SYSTEM. THIS SNAPSHOT COPY IS CALLED THE RESTARTING POINT OF THE FILE SYSTEM.
574
REMOUNT SECONDARY FILE SYSTEM AS READ/WRITE
END

FIG. 33

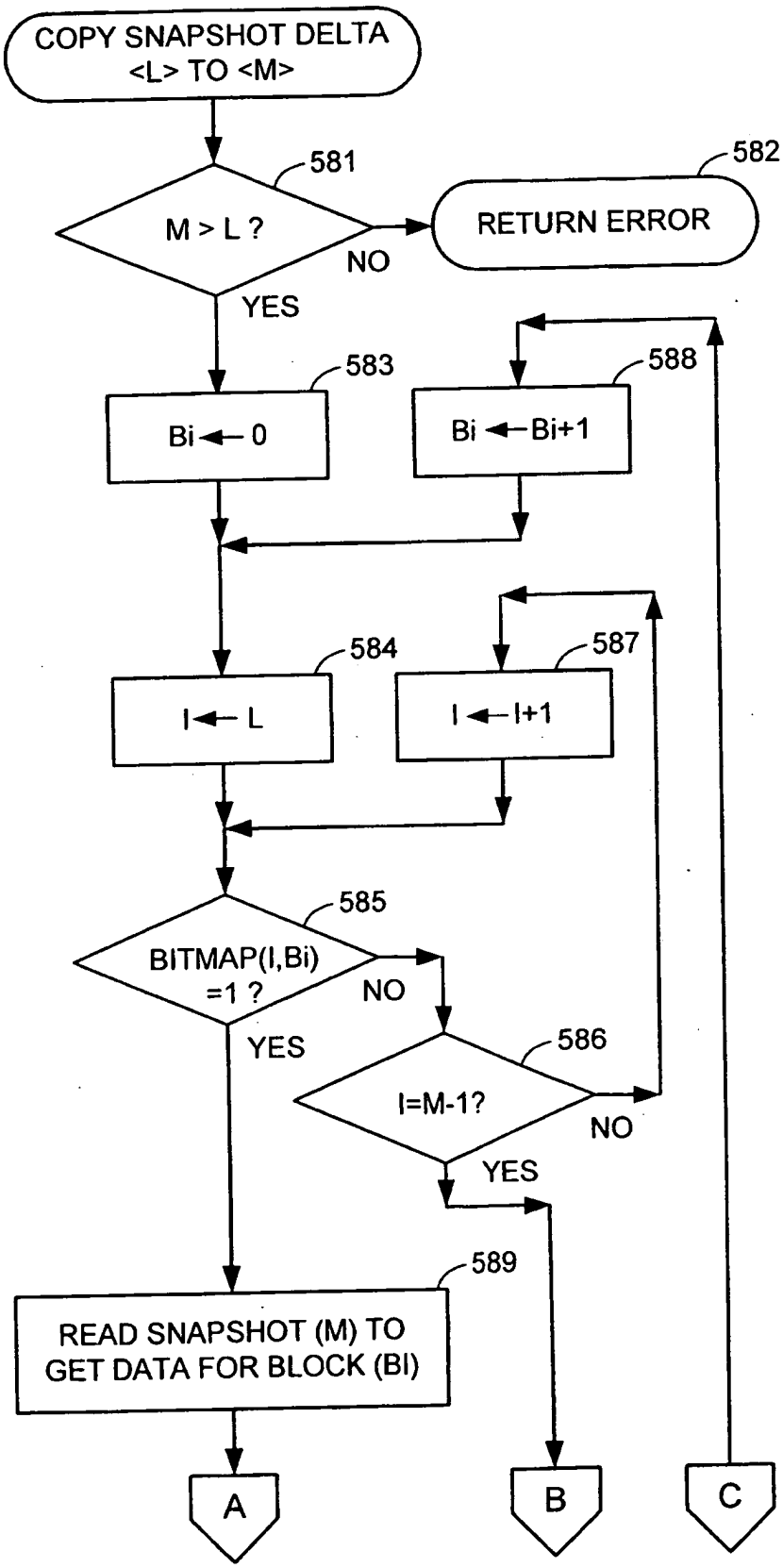

COPY SNAPSHOT DELTA <L> TO <M>
581
M > L ?
NO
582
RETURN ERROR
YES
583
Bi ← 0
588
Bi ← Bi+1
584
I ← L
587
I ← I+1
585
BITMAP(I,Bi) =1 ?
NO
YES
586
I=M-1?
NO
YES
589
READ SNAPSHOT (M) TO GET DATA FOR BLOCK (BI)
A
B
C

FIG. 34

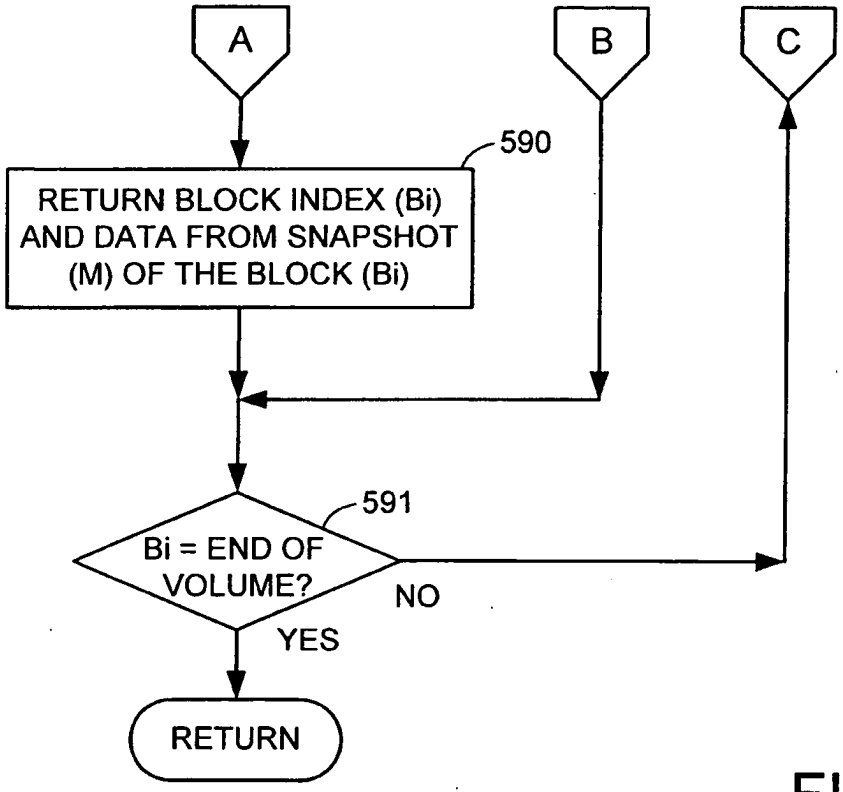

A
B
C
590
RETURN BLOCK INDEX (Bi) AND DATA FROM SNAPSHOT (M) OF THE BLOCK (Bi)
591
Bi = END OF VOLUME?
NO
YES
RETURN

FIG. 35

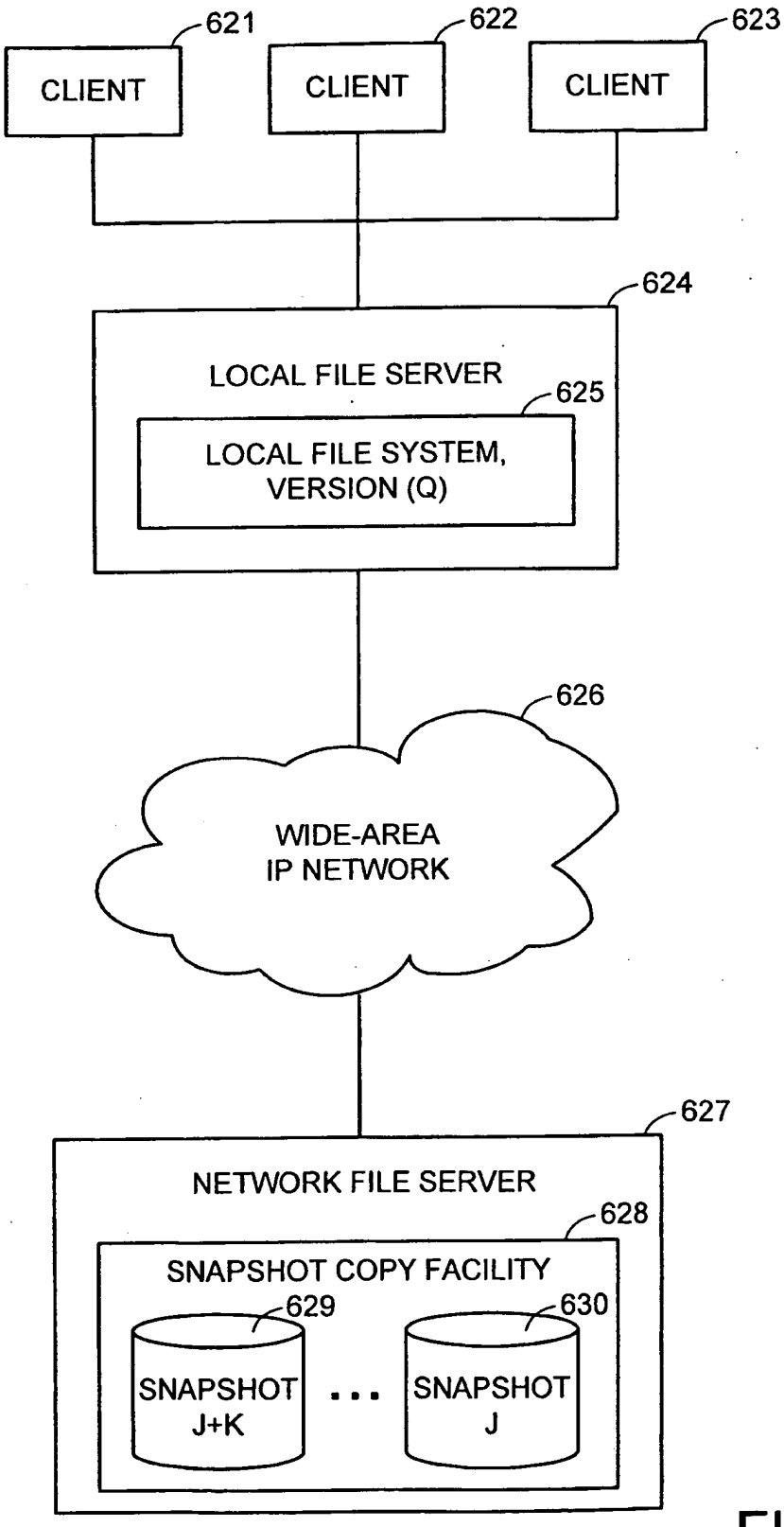

621
CLIENT
622
CLIENT
623
CLIENT
624
LOCAL FILE SERVER
625
LOCAL FILE SYSTEM,
VERSION (Q)
626
WIDE-AREA
IP NETWORK
627
NETWORK FILE SERVER
628
SNAPSHOT COPY FACILITY
629
SNAPSHOT
J+K
...
630
SNAPSHOT
J

FIG. 36

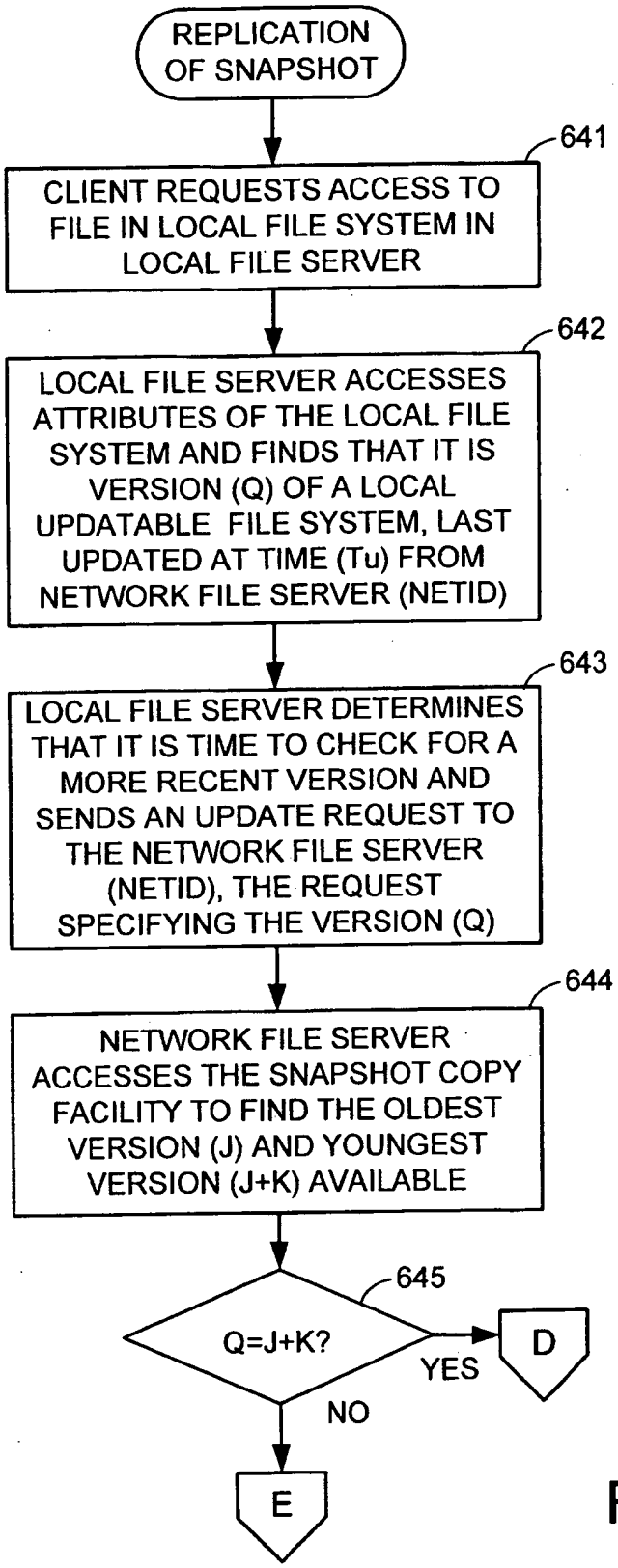

REPLICATION OF SNAPSHOT
641
CLIENT REQUESTS ACCESS TO FILE IN LOCAL FILE SYSTEM IN LOCAL FILE SERVER
642
LOCAL FILE SERVER ACCESSES ATTRIBUTES OF THE LOCAL FILE SYSTEM AND FINDS THAT IT IS VERSION (Q) OF A LOCAL UPDATABLE FILE SYSTEM, LAST UPDATED AT TIME (Tu) FROM NETWORK FILE SERVER (NETID)
643
LOCAL FILE SERVER DETERMINES THAT IT IS TIME TO CHECK FOR A MORE RECENT VERSION AND SENDS AN UPDATE REQUEST TO THE NETWORK FILE SERVER (NETID), THE REQUEST SPECIFYING THE VERSION (Q)
644
NETWORK FILE SERVER ACCESSES THE SNAPSHOT COPY FACILITY TO FIND THE OLDEST VERSION (J) AND YOUNGEST VERSION (J+K) AVAILABLE
645
Q=J+K?
YES
D
NO
E

FIG. 37

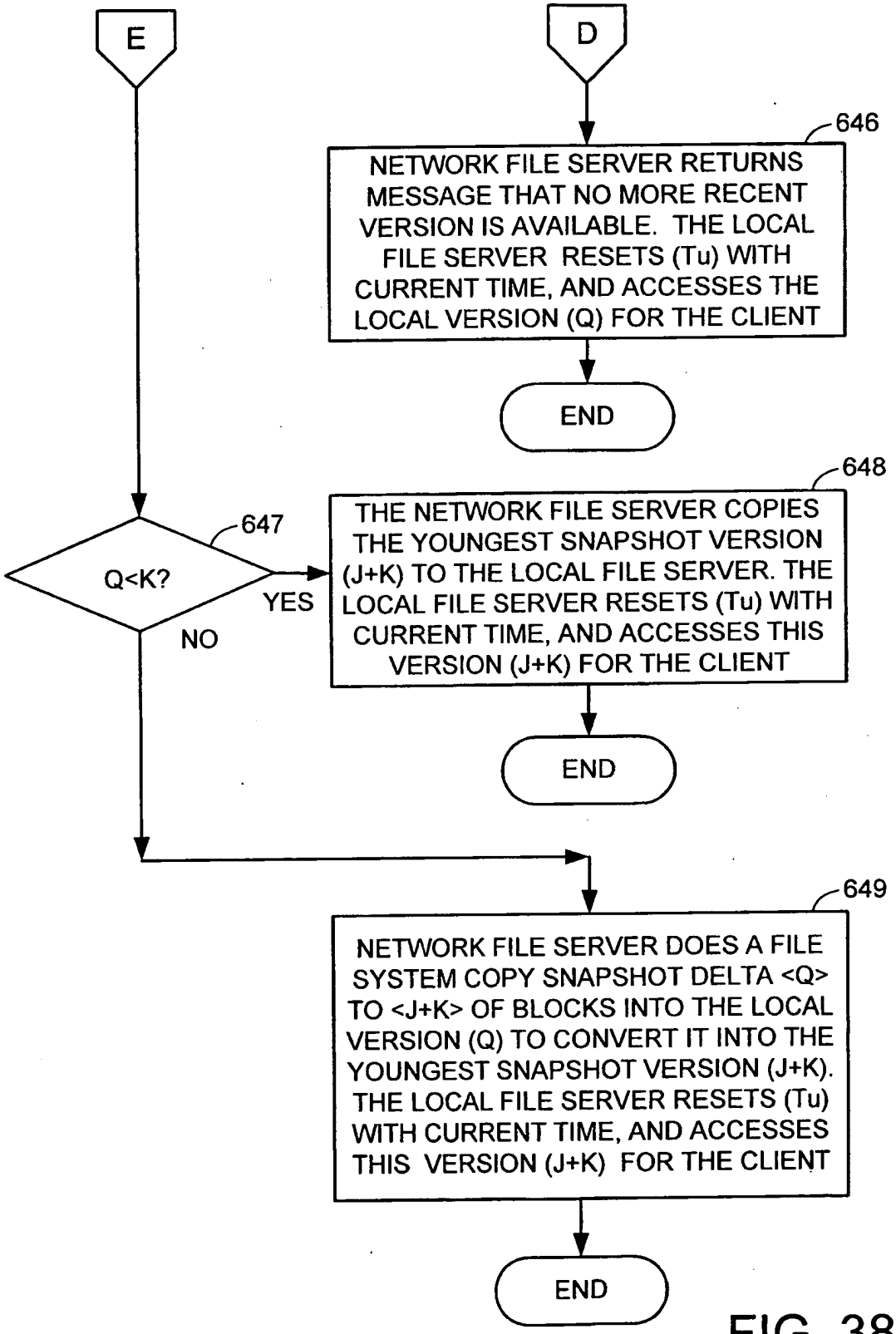

E
D
646
NETWORK FILE SERVER RETURNS MESSAGE THAT NO MORE RECENT VERSION IS AVAILABLE. THE LOCAL FILE SERVER RESETS (Tu) WITH CURRENT TIME, AND ACCESSES THE LOCAL VERSION (Q) FOR THE CLIENT
END
648
647
Q<K?
YES
NO
THE NETWORK FILE SERVER COPIES THE YOUNGEST SNAPSHOT VERSION (J+K) TO THE LOCAL FILE SERVER. THE LOCAL FILE SERVER RESETS (Tu) WITH CURRENT TIME, AND ACCESSES THIS VERSION (J+K) FOR THE CLIENT
END
649
NETWORK FILE SERVER DOES A FILE SYSTEM COPY SNAPSHOT DELTA <Q> TO <J+K> OF BLOCKS INTO THE LOCAL VERSION (Q) TO CONVERT IT INTO THE YOUNGEST SNAPSHOT VERSION (J+K). THE LOCAL FILE SERVER RESETS (Tu) WITH CURRENT TIME, AND ACCESSES THIS VERSION (J+K) FOR THE CLIENT
END

FIG. 38

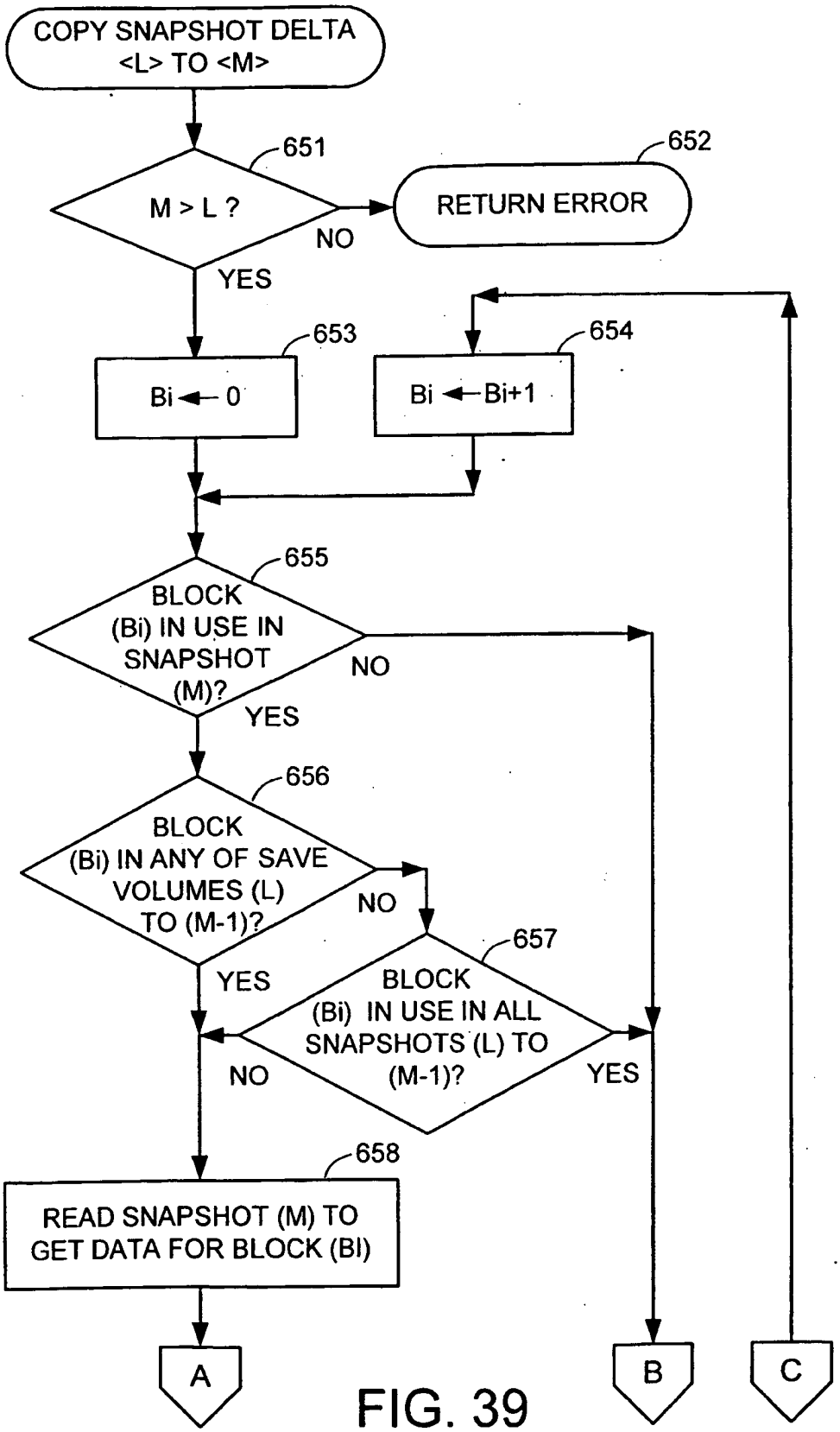

COPY SNAPSHOT DELTA <L> TO <M>
651
M > L ?
NO
652
RETURN ERROR
YES
653
Bi ← 0
654
Bi ← Bi+1
655
BLOCK (Bi) IN USE IN SNAPSHOT (M)?
NO
YES
656
BLOCK (Bi) IN ANY OF SAVE VOLUMES (L) TO (M-1)?
NO
YES
657
BLOCK (Bi) IN USE IN ALL SNAPSHOTS (L) TO (M-1)?
NO
YES
658
READ SNAPSHOT (M) TO GET DATA FOR BLOCK (BI)
A
B
C

FIG. 39

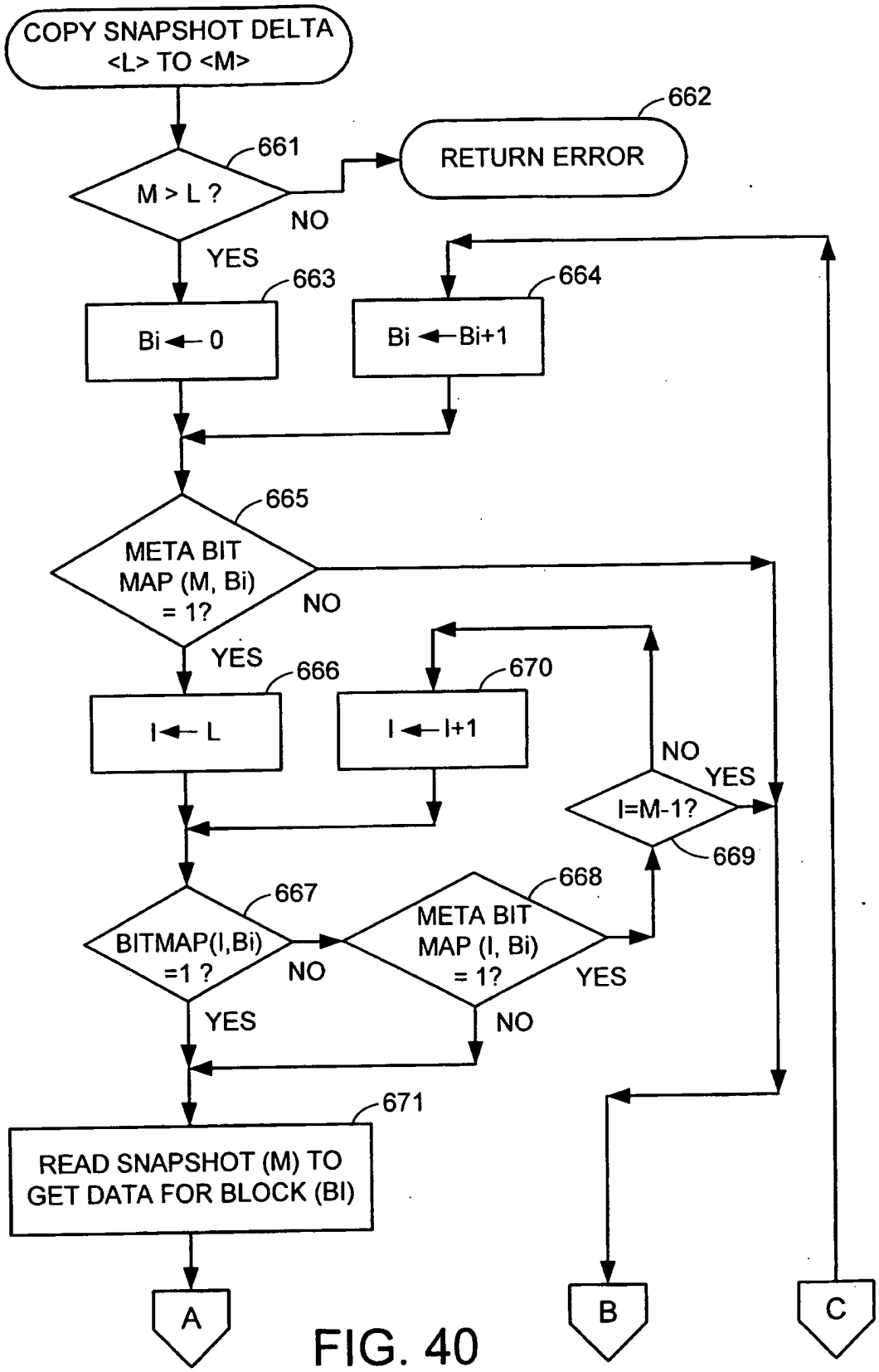

COPY SNAPSHOT DELTA <L> TO <M>
661
M > L ?
NO
662
RETURN ERROR
YES
663
Bi ← 0
664
Bi ← Bi+1
665
META BIT MAP (M, Bi) = 1?
NO
YES
666
I ← L
670
I ← I+1
NO
YES
I=M-1?
669
667
BITMAP(I,Bi) =1 ?
NO
668
META BIT MAP (I, Bi) = 1?
YES
YES
NO
671
READ SNAPSHOT (M) TO GET DATA FOR BLOCK (BI)
A
B
C

FIG. 40

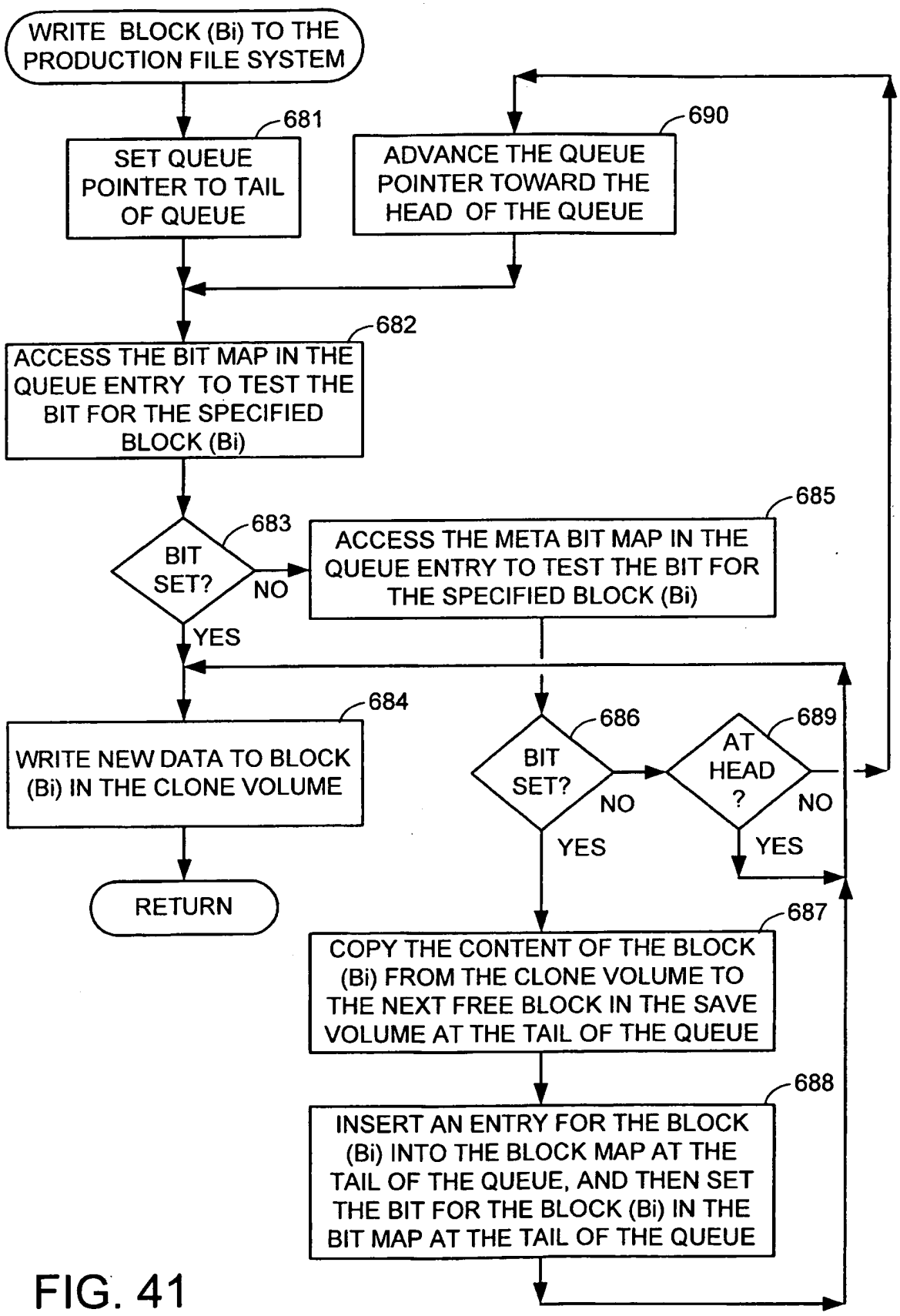

WRITE BLOCK (Bi) TO THE PRODUCTION FILE SYSTEM
681
SET QUEUE POINTER TO TAIL OF QUEUE
690
ADVANCE THE QUEUE POINTER TOWARD THE HEAD OF THE QUEUE
682
ACCESS THE BIT MAP IN THE QUEUE ENTRY TO TEST THE BIT FOR THE SPECIFIED BLOCK (Bi)
683
BIT SET?
NO
YES
685
ACCESS THE META BIT MAP IN THE QUEUE ENTRY TO TEST THE BIT FOR THE SPECIFIED BLOCK (Bi)
684
WRITE NEW DATA TO BLOCK (Bi) IN THE CLONE VOLUME
RETURN
686
BIT SET?
NO
YES
689
AT HEAD ?
NO
YES
687
COPY THE CONTENT OF THE BLOCK (Bi) FROM THE CLONE VOLUME TO THE NEXT FREE BLOCK IN THE SAVE VOLUME AT THE TAIL OF THE QUEUE
688
INSERT AN ENTRY FOR THE BLOCK (Bi) INTO THE BLOCK MAP AT THE TAIL OF THE QUEUE, AND THEN SET THE BIT FOR THE BLOCK (Bi) IN THE BIT MAP AT THE TAIL OF THE QUEUE

FIG. 41

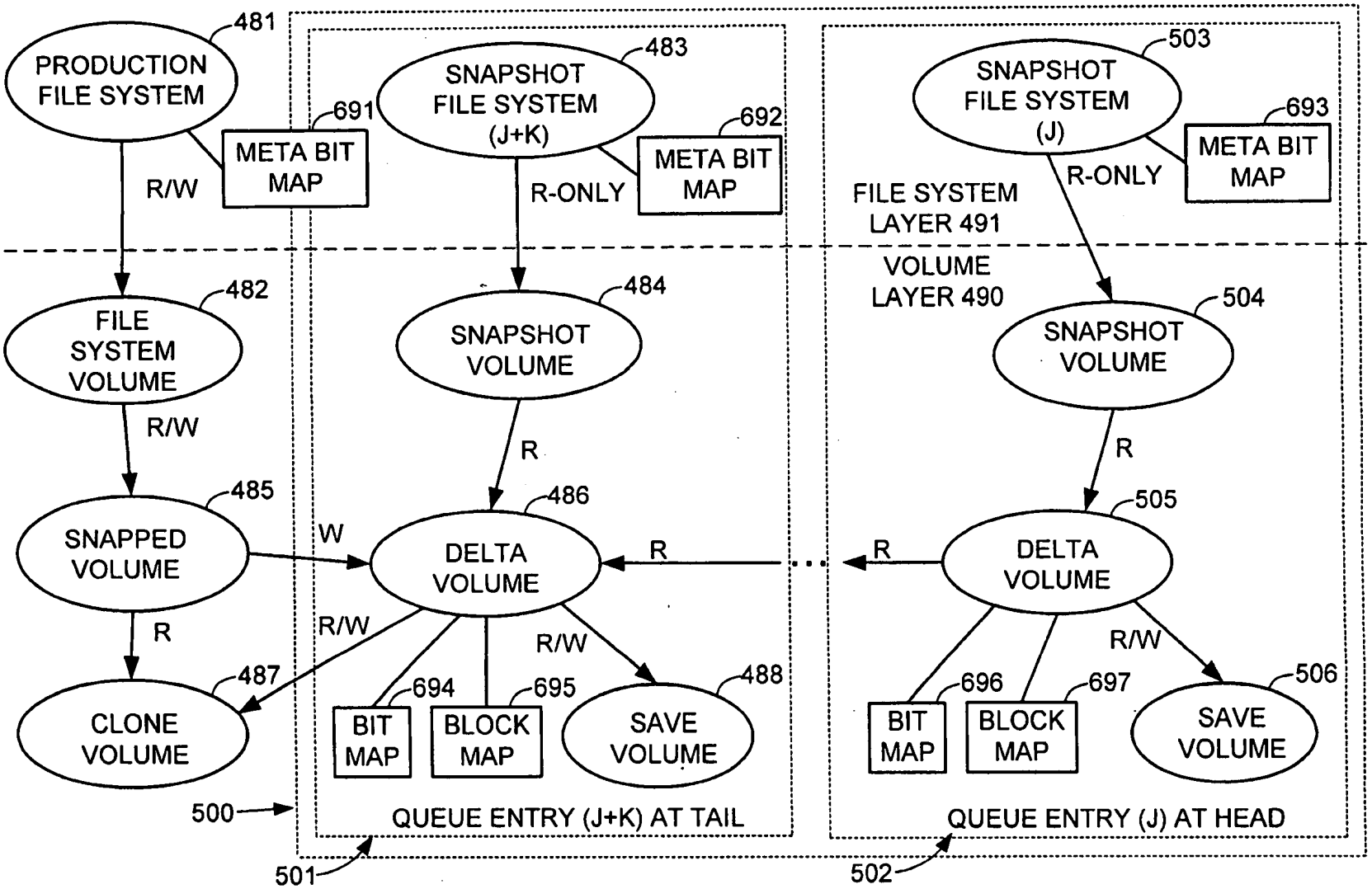

481
PRODUCTION FILE SYSTEM
691
META BIT MAP
R/W
483
SNAPSHOT FILE SYSTEM (J+K)
692
META BIT MAP
R-ONLY
503
SNAPSHOT FILE SYSTEM (J)
693
META BIT MAP
R-ONLY
FILE SYSTEM LAYER 491
VOLUME LAYER 490
482
FILE SYSTEM VOLUME
R/W
484
SNAPSHOT VOLUME
R
504
SNAPSHOT VOLUME
R
485
SNAPPED VOLUME
W
486
DELTA VOLUME
R
R
505
DELTA VOLUME
R
R/W
R/W
R/W
487
CLONE VOLUME
694
BIT MAP
695
BLOCK MAP
488
SAVE VOLUME
696
BIT MAP
697
BLOCK MAP
506
SAVE VOLUME
500
QUEUE ENTRY (J+K) AT TAIL
QUEUE ENTRY (J) AT HEAD
501
502

FIG. 42

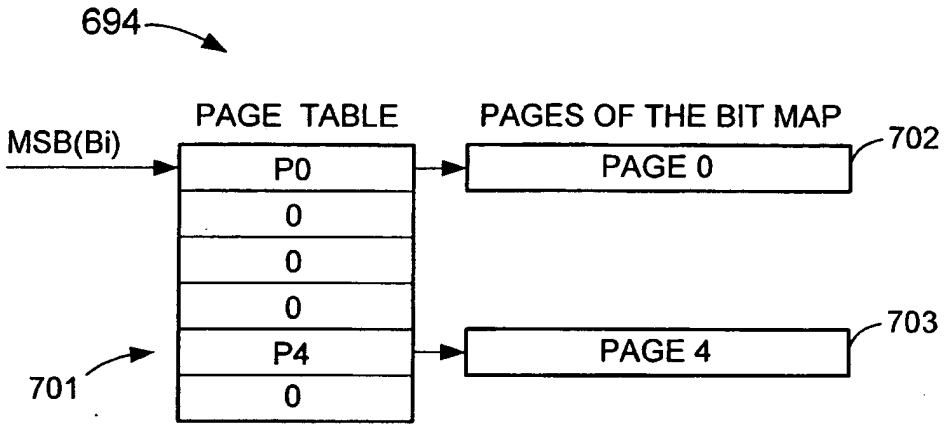

694
PAGE TABLE
PAGES OF THE BIT MAP
MSB(Bi)
P0
PAGE 0
702
0
0
0
P4
PAGE 4
703
0
701

FIG. 43

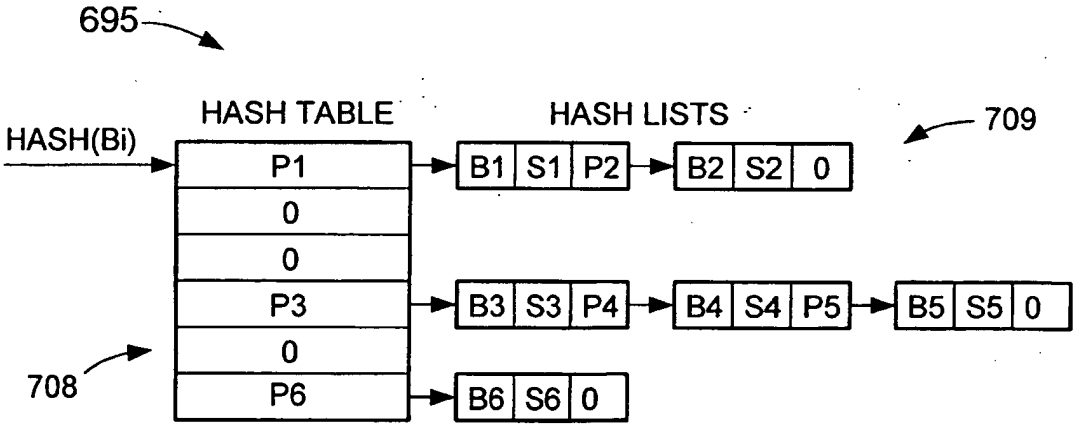

695
HASH TABLE
HASH LISTS
709
HASH(Bi)
P1
B1 S1 P2
B2 S2 0
0
0
P3
B3 S3 P4
B4 S4 P5
B5 S5 0
0
708
P6
B6 S6 0

FIG. 44

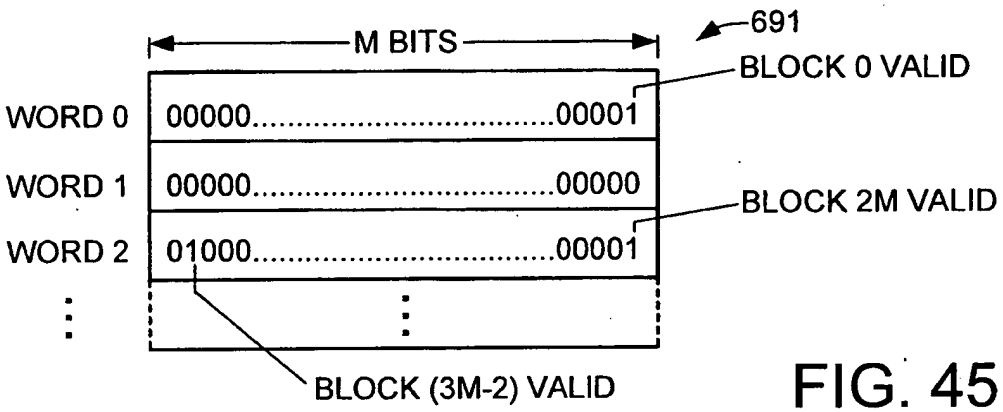

691
M BITS
BLOCK 0 VALID
WORD 0
00000.....00001
WORD 1
00000.....00000
BLOCK 2M VALID
WORD 2
01000.....00001
BLOCK (3M-2) VALID

FIG. 45

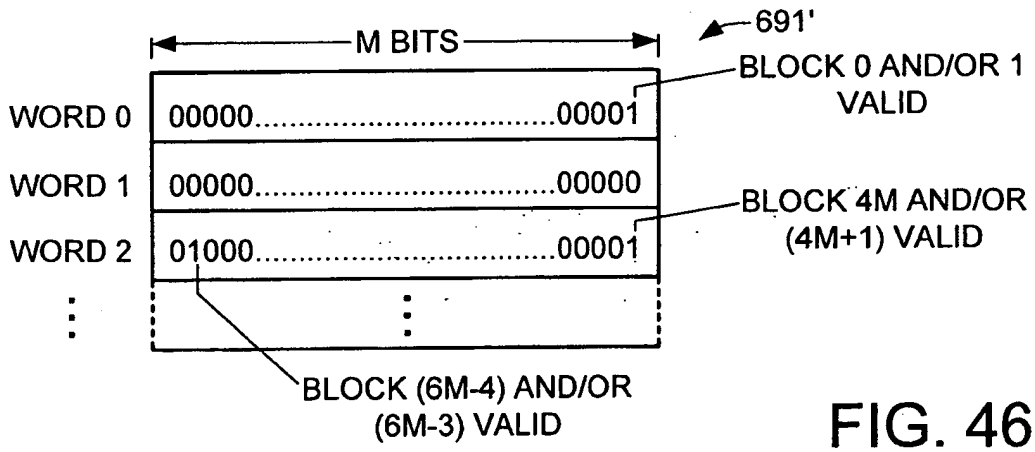

691'
M BITS
BLOCK 0 AND/OR 1
VALID
WORD 0
00000.....00001
WORD 1
00000.....00000
BLOCK 4M AND/OR
(4M+1) VALID
WORD 2
01000.....00001
BLOCK (6M-4) AND/OR
(6M-3) VALID

FIG. 46

US 20040163009A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2004/0163009 A1

Goldstein et al. (43) Pub. Date: Aug. 19, 2004

(54) PHYSICAL INCREMENTAL BACKUP USING SNAPSHOTS

(76) Inventors: **Andrew C. Goldstein**, Hudson, MA (US); **David W. Thiel**, Colorado Springs, CO (US); **Richard F. Wrenn**, Colorado Springs, CO (US)

Correspondence Address:
HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P. O. Box 272400
Fort Collins, CO 80527-2400 (US)

(21) Appl. No.: **10/700,140**

(22) Filed: **Nov. 3, 2003**

Related U.S. Application Data

(63) Continuation of application No. 09/599,707, filed on Jun. 22, 2000, now Pat. No. 6,665,815.

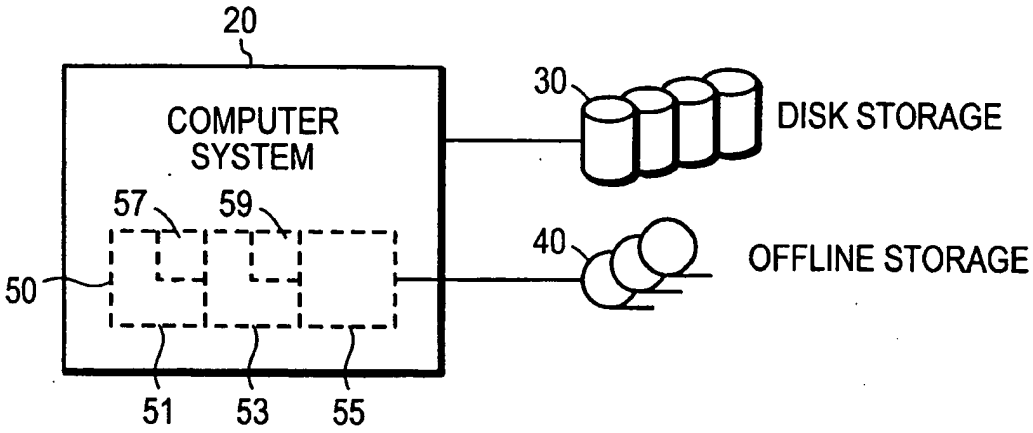
Publication Classification

(51) **Int. Cl.**[7] **H02H 3/05**

(52) **U.S. Cl.** **714/6**

(57) **ABSTRACT**

A backup apparatus and method suitable for protecting the data volume in a computer system function by acquiring a base state snapshot and a sequential series of data volume snapshots, the apparatus concurrently generating succedent and precedent lists of snapshot differences which are used to create succedent and precedent backups respectively. The data volume is restored by overwriting the base state data with data blocks identified in one or more succedent backups. File recovery is accomplished by overwriting data from a current snapshot with one or more precedent backups.



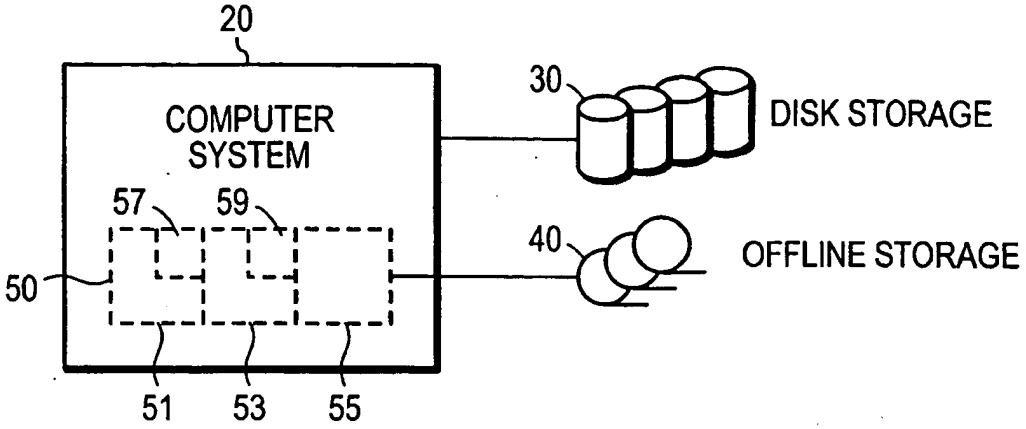

20
COMPUTER SYSTEM
57
59
50
51
53
55
30
DISK STORAGE
40
OFFLINE STORAGE

FIG. 1

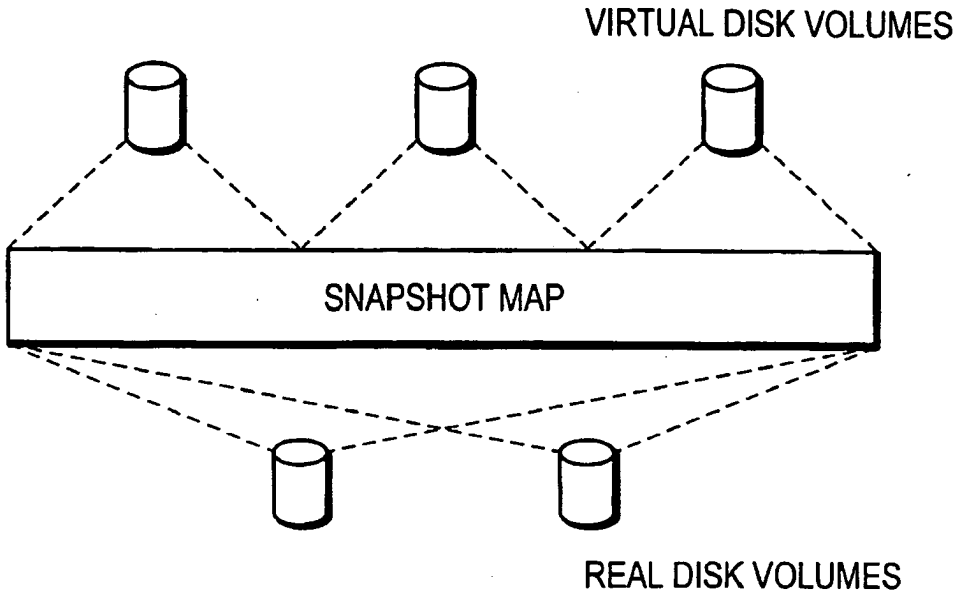


FIG. 2

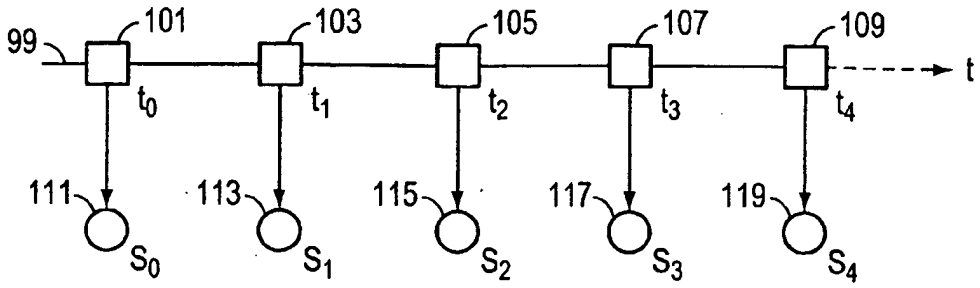

99
101
103
105
107
109
t
t_0
t_1
t_2
t_3
t_4
111
113
115
117
119
S_0
S_1
S_2
S_3
S_4

FIG. 3

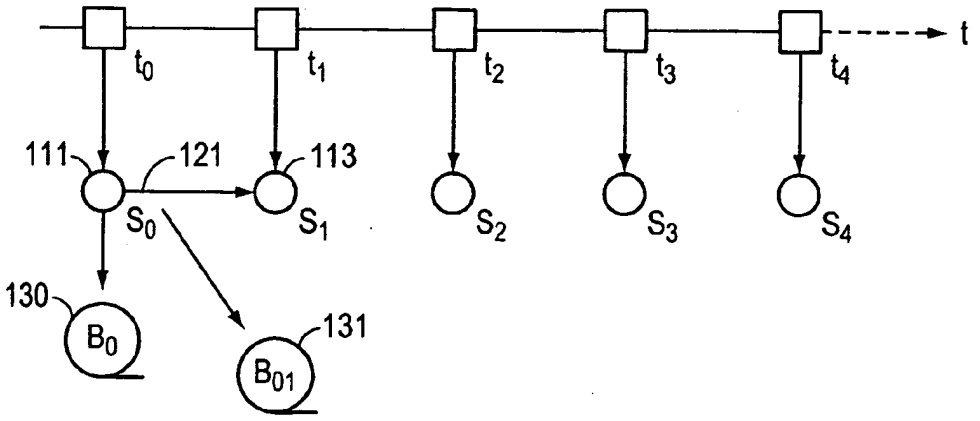

t
t_0
t_1
t_2
t_3
t_4
111
121
113
S_0
S_1
S_2
S_3
S_4
130
131
B_0
B_{01}

FIG. 4

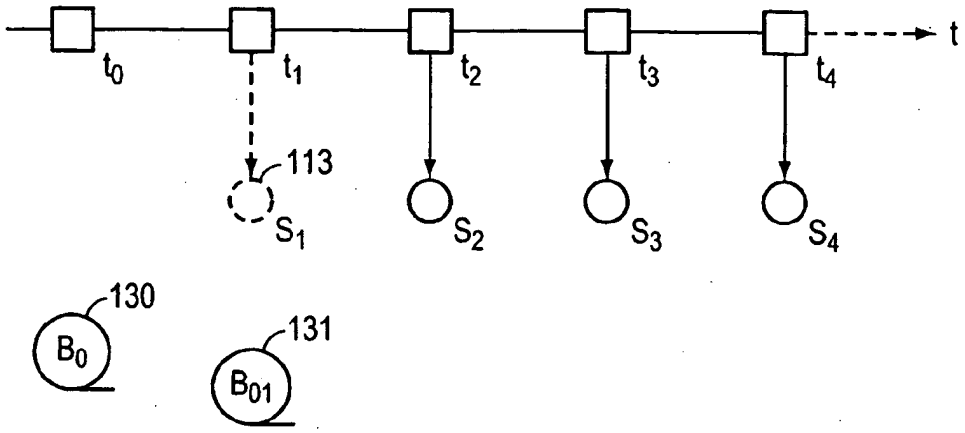

t
t_0
t_1
t_2
t_3
t_4
113
S_1
S_2
S_3
S_4
130
B_0
131
B_{01}

FIG. 5

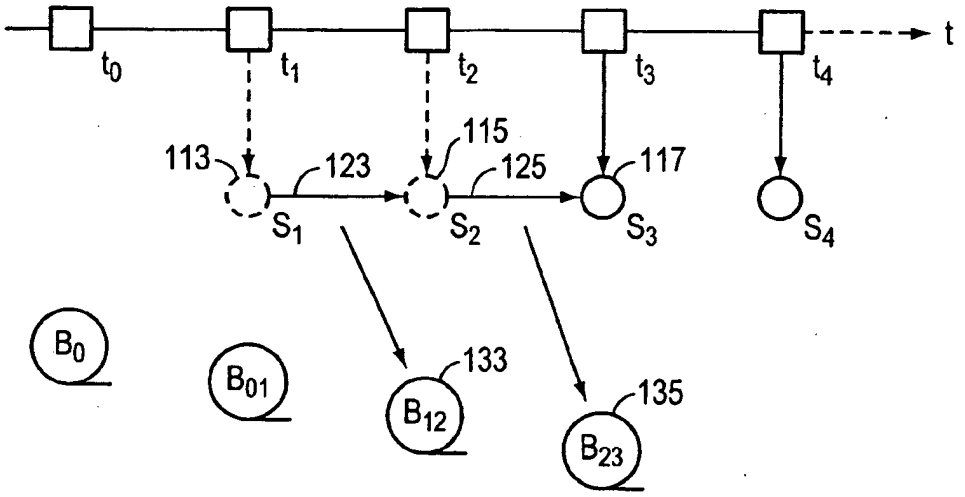

t
t_0
t_1
t_2
t_3
t_4
113
123
115
125
117
S_1
S_2
S_3
S_4
B_0
B_{01}
133
B_{12}
135
B_{23}

FIG. 6

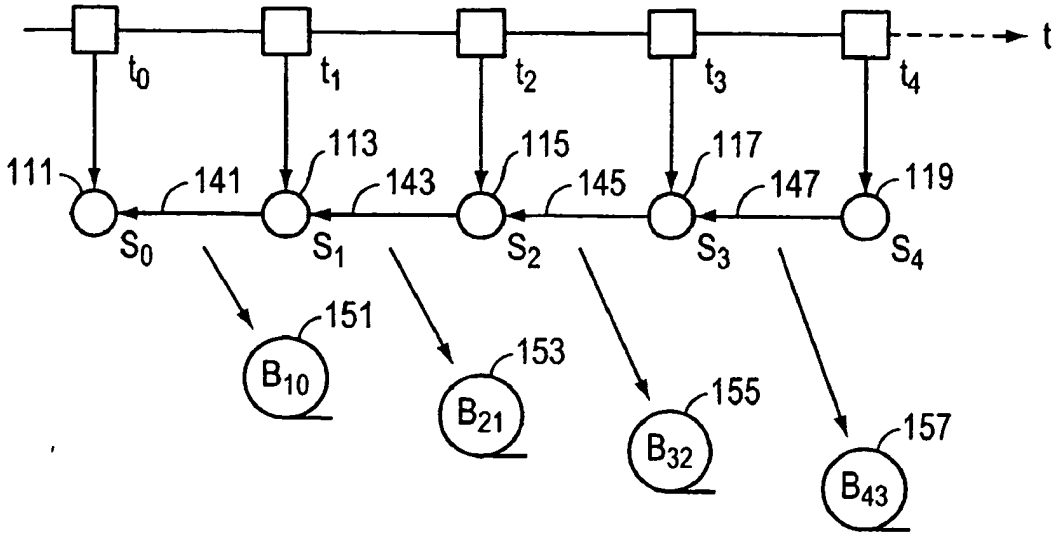

t
t_0
t_1
t_2
t_3
t_4
111
141
113
143
115
145
117
147
119
S_0
S_1
S_2
S_3
S_4
151
B_{10}
153
B_{21}
155
B_{32}
157
B_{43}

FIG. 7

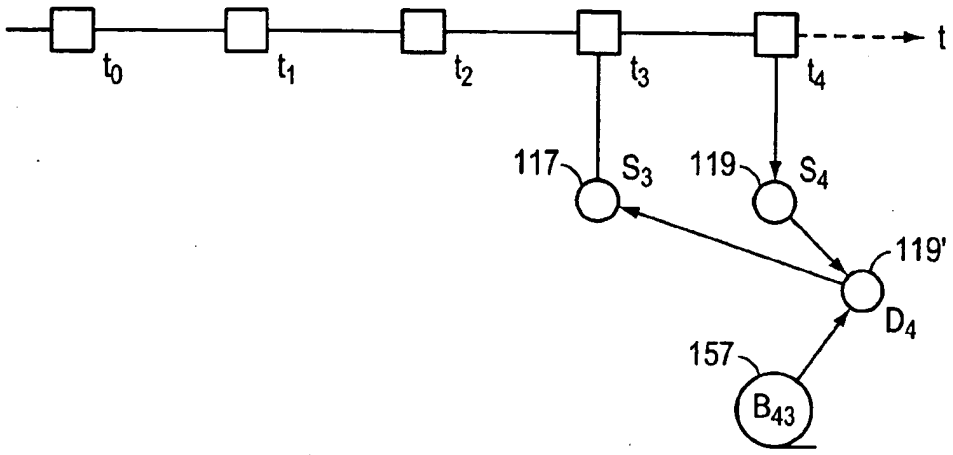

t
t_0
t_1
t_2
t_3
t_4
117
S_3
119
S_4
119'
D_4
157
B_{43}

FIG. 8

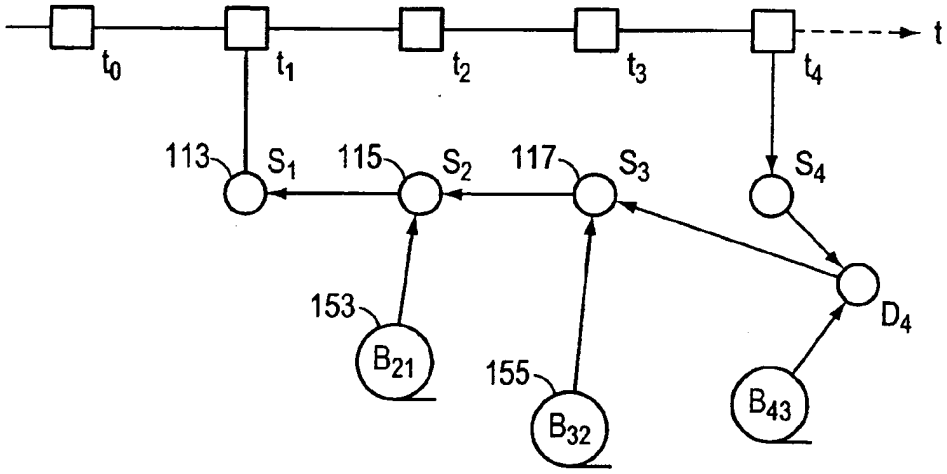

t
t_0
t_1
t_2
t_3
t_4
113
S_1
115
S_2
117
S_3
S_4
153
B_{21}
155
B_{32}
D_4
B_{43}

FIG. 9

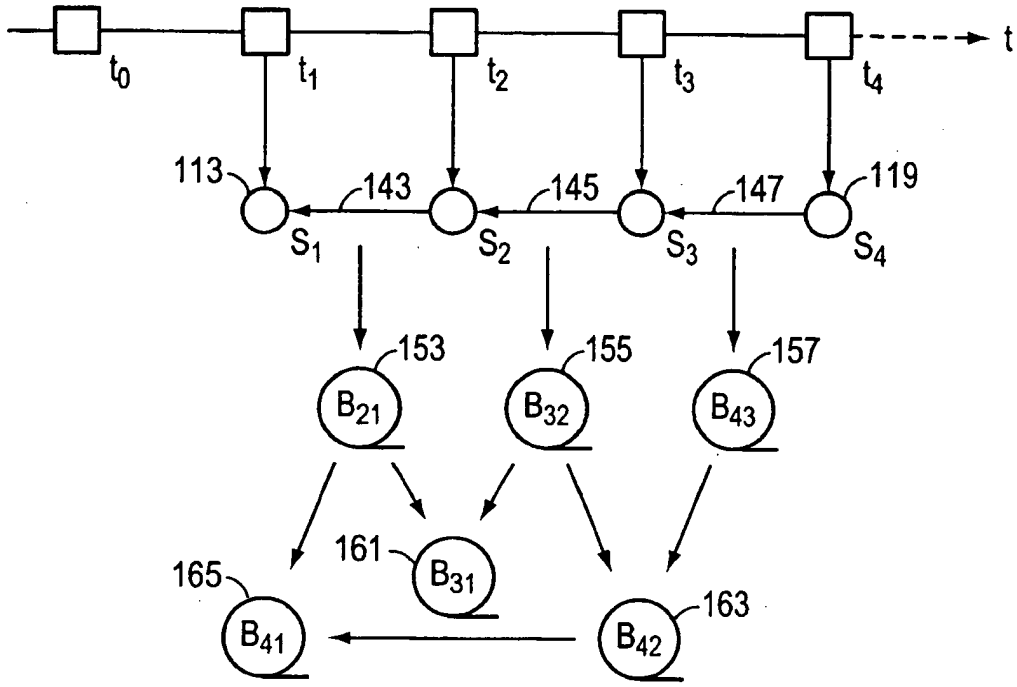

t
t_0
t_1
t_2
t_3
t_4
113
143
145
147
119
S_1
S_2
S_3
S_4
153
155
157
B_{21}
B_{32}
B_{43}
161
B_{31}
165
B_{41}
B_{42}
163

FIG. 10

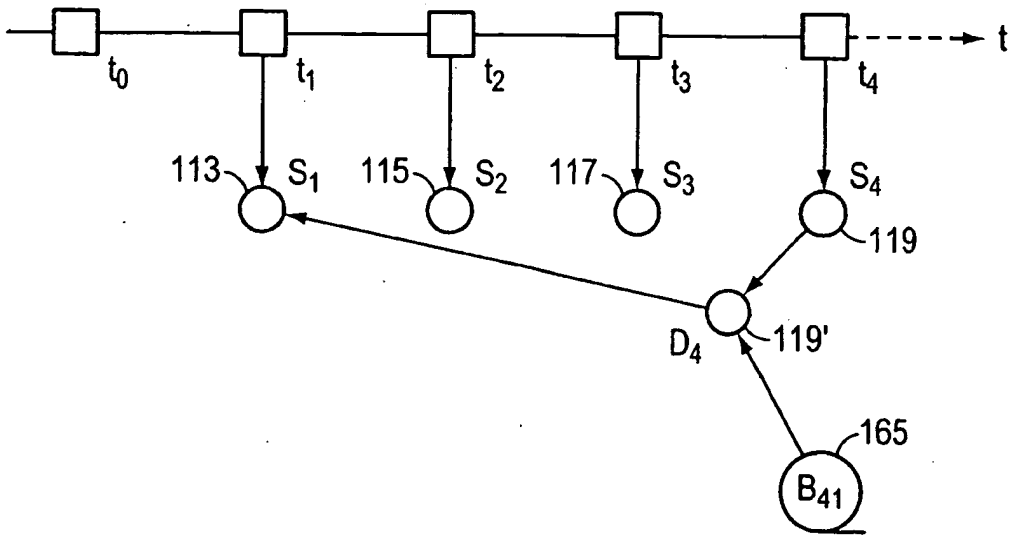

t
t_0
t_1
t_2
t_3
t_4
113
S_1
115
S_2
117
S_3
S_4
119
D_4
119'
165
B_{41}

FIG. 11

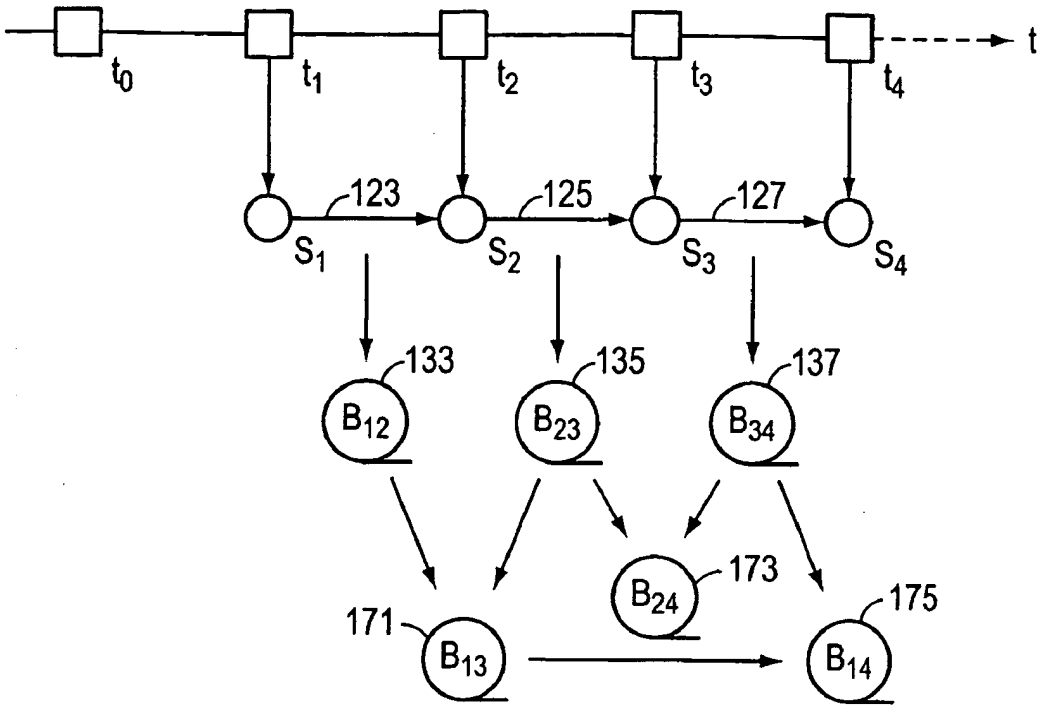

t
t_0
t_1
t_2
t_3
t_4
123
125
127
S_1
S_2
S_3
S_4
133
135
137
B_{12}
B_{23}
B_{34}
173
175
171
B_{24}
B_{13}
B_{14}

FIG. 12

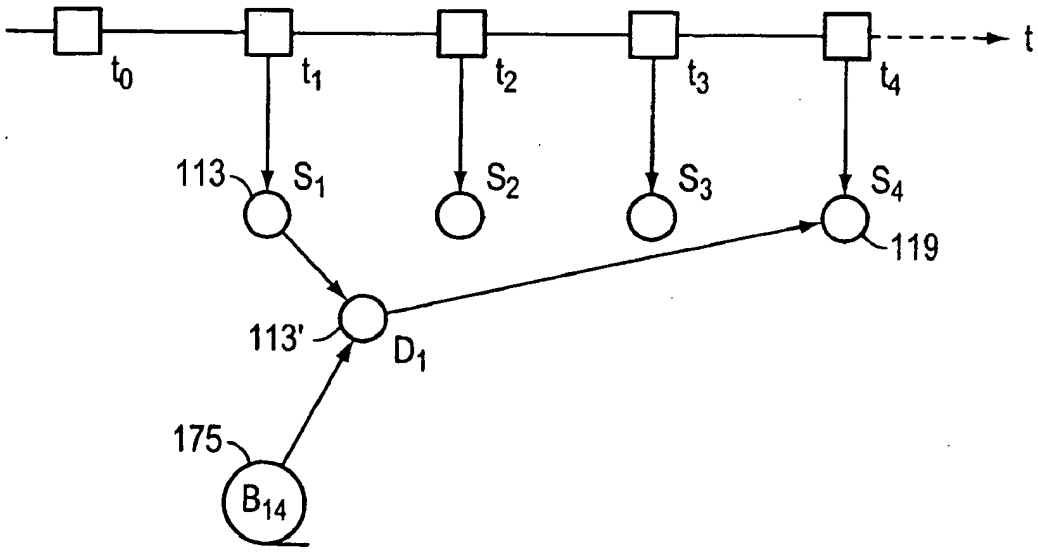

t
t_0
t_1
t_2
t_3
t_4
113
S_1
S_2
S_3
S_4
119
113'
D_1
175
B_{14}

FIG. 13

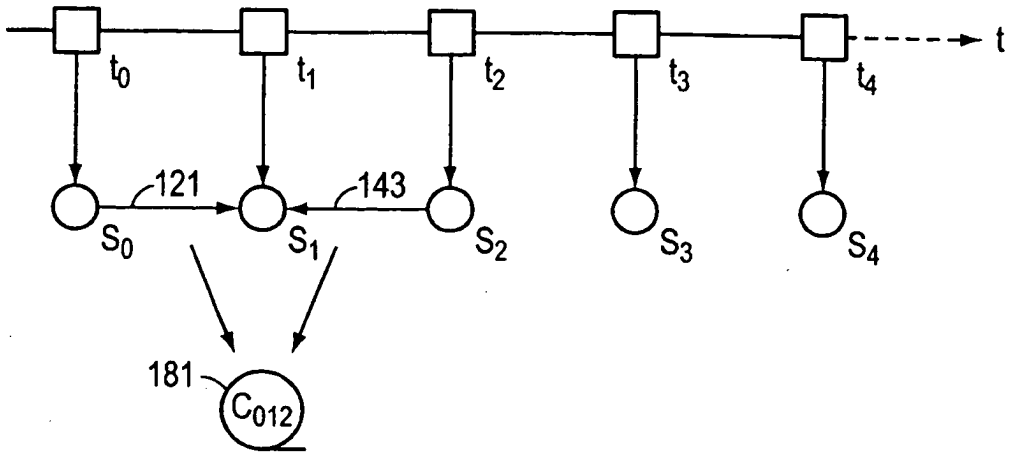

t
t_0
t_1
t_2
t_3
t_4
121
143
S_0
S_1
S_2
S_3
S_4
181
C_{012}

FIG. 14

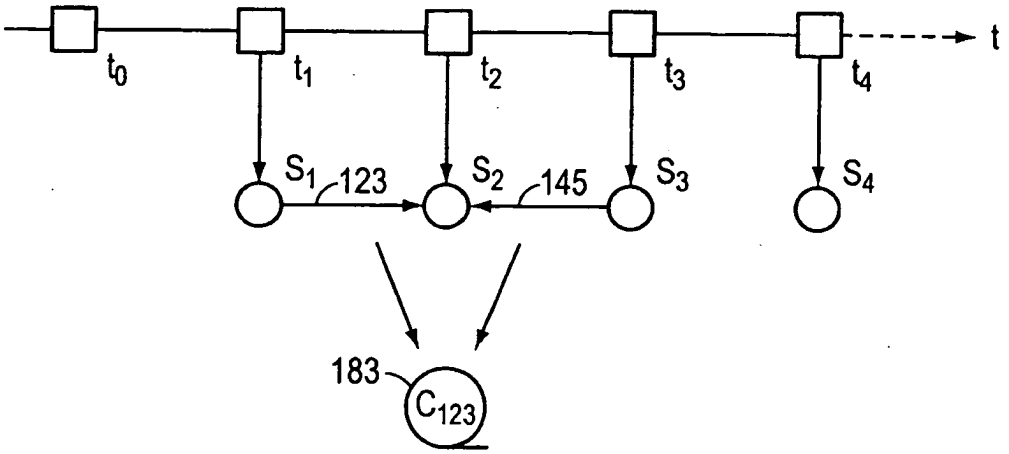

t
t_0
t_1
t_2
t_3
t_4
S_1
123
S_2
145
S_3
S_4
183
C_{123}

FIG. 15

PHYSICAL INCREMENTAL BACKUP USING SNAPSHOTS

BACKGROUND OF THE INVENTION

[0001] 1. Field of the Invention .

[0002] The present invention is related to the protection of computer data and, in particular, to a system and method for making backups made to offline storage media, such as tapes, that are not directly accessible as file-structured devices.

[0003] 2. Description of the Prior Art

[0004] A backup procedure is a function that is typically included in most computer operating system software. One of the most pressing backup problems over the last several years has been the time vs. volume dilemma. Storage capacity and actual online storage volumes have increased at a geometric rate, roughly doubling every two years. However, the bandwidth of storage subsystems, that is, the rate at which data can be transferred into and out of the storage subsystem, has increased at a much slower rate. Consequently, the time required to make a complete copy of online storage has steadily increased. In addition, most backup procedures use the file system to produce file-coherent backups. This imposes additional overhead that considerably reduces the effective bandwidth of the storage subsystem. Many hours are required to make a full backup of a large scale installation.

[0005] At the same time, many computer installations are faced with increasingly stringent uptime requirements. The 'backup window' (i.e., the time during which the data is stable so that a coherent backup can be made) continues to shrink. In many cases, the available backup window is already smaller than the time required to create a full backup. Computer installations have applied a number of ad hoc measures to address these difficulties, with varying degrees of success. Consequently, many installations are running with inadequate or no backup coverage because the backup window is inadequate.

[0006] One approach is the physical backup, a brute force approach that copies the disk volume block for block, ignoring the file structure. The physical backup can operate at the maximum possible data rate of the storage subsystem. However, the physical backup does suffer from certain disadvantages. First, all activity on the disk volume must be completely frozen for the backup to be useful because there is no coordination with the file system. Second, recovery of individual files from a physical backup is cumbersome because the entire backup must be restored to disk to process the file structure. Third, even the maximum storage bandwidth may be inadequate in a very large-scale storage environment to perform a full physical backup in the available backup window.

[0007] Another well-known approach is the incremental file backup. In this approach, individual files are backed up if they have been modified since the previous backup. If they have not changed, they are not backed up. This method reduces the volume of data to be backed up to the volume of files that have changed. It works well in an environment where files are relatively small and are typically modified in their entirety. It does not work well when files are large, and typical updates modify a small part of the file, because even with a small modification the entire file must be backed up. Also, complete reconstruction of a data volume from incremental file backups can be problematical because files that are deleted during the life of the volume will reappear when successive incremental backups are restored. Depending on the design of the file system and the backup, incremental restores can introduce other inaccuracies, compared to the original volume.

[0008] However, the basic incremental backup method suffers from the disadvantages that a considerable amount of time is spent processing the file structure to locate files that need to be backed up, and the process of reconstructing a disk volume from incremental backups is complex and trouble-prone. Accordingly, the system manager would typically perform periodic full backups in addition to the incremental backups to limit the risk of recovering with incremental backups alone.

[0009] Another approach is disclosed in U.S. Pat. No. 5,835,953, "Backup system that takes a snapshot of the locations in a mass storage device that has been identified for updating prior to updating," issued to Ohran. This basic incremental backup method includes maintaining a "virtual disk" subsystem capable of generating snapshots and making a full copy of the snapshot for remote disk storage. An initial and a subsequent snapshots are obtained. Snapshot mapping data is used to determine the data blocks which have changed from the initial snapshot to the subsequent snapshot. The changed blocks are then copied to the remote storage, the initial snapshot is deleted, and the process is continued as needed.

[0010] The method disclosed in Ohran '953, for example, provides a complete backup copy of the data volume to allow recovery if the original volume is lost. However, the prior art does not address situations in which individual files need to be recovered, such as when a file is erroneously deleted or when an application fails and writes incorrect data. Once a snapshot and copy cycle have been performed using a conventional method, the previous (and possible the only valid) file contents are lost. Thus, there is a need in the art for an effective backup strategy which preserves old versions of the file contents at is suitable intervals to allow recovery when errors are subsequently detected.

SUMMARY OF THE INVENTION

[0011] The data volume in a computer system can be protected by first acquiring a base state snapshot and a subsequent series of data volume snapshots. A plurality of snapshot difference lists can be generated by identifying those data blocks which differ between sequential snapshots. A precedent snapshot difference list, generated by identifying the data blocks in any snapshot differing from the data blocks in a subsequent snapshot, is used to recover files without incurring a full restore. The data blocks described by the snapshot difference list are copied to backup storage and the snapshot is deleted. File recovery is accomplished by overwriting data from a current snapshot list with one or more precedent backups. A succedent snapshot difference list, generated by identifying the data blocks in any snapshot differing from the data blocks in a previous snapshot, is used to restore a data volume. The data volume is restored by restoring the base state data with data blocks contained in one or more succedent backups.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] The invention description below refers to the accompanying drawings, of which:

[0013] **FIG. 1** is a diagrammatical illustration of an apparatus for protecting the data volume in a computer system, in accordance with the present invention;

[0014] **FIG. 2** is a diagrammatical illustration of the relationship between real disk volumes and virtual disk volumes;

[0015] **FIG. 3** is a timeline showing a series of snapshots being acquired of a sequence of data volume consistent states;

[0016] FIGS. **4-6** show the generation of a base state backup and first through third succedent backups on the timeline of **FIG. 3**;

[0017] **FIG. 7** shows the generation of first through fourth precedent backups on the timeline of **FIG. 3**;

[0018] FIGS. **8-9** illustrate the restoration of a file by using the precedent backups generated in **FIG. 7**;

[0019] **FIG. 10** shows the process of concatenating two or more of the precedent backups of **FIG. 7** to form one or more concatenated precedent backups;

[0020] **FIG. 11** illustrates the restoration of a file using one of the concatenated backups of **FIG. 10**;

[0021] **FIG. 12** shows the process of concatenating two or more of the succedent backups of **FIG. 6** to form one or more concatenated succedent backups;

[0022] **FIG. 13** illustrates a method of disaster recovery using one of the concatenated backups of **FIG. 12**; and,

[0023] FIGS. **14-15** illustrate the generation of composite backups.

DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

[0024] **FIG. 1** is a diagrammatical illustration of an apparatus for protecting the data volume in a computer system **20**, in accordance with the present invention. The computer system **20** accesses and stores the data volume in a disk storage **30**. The computer system **20** includes backup processing means **50** for creating backups of portions of the disk storage **30**. Backup processing means **50** includes a snapshot section **51** for acquiring snapshots **57** of the consistent states of the data volume in disk storage **30**, as explained in greater detail below. The snapshots **57** are compared by a processing unit **53**, as explained in greater detail below, to produce a list of blocks that have changed between the snapshots **57** so that those blocks may be copied into backups **59**. In a preferred embodiment, backup processing means **50** also includes a sending unit **55** for storing the backups **59** in the offline storage **40**.

[0025] Referring to **FIG. 2**, a snapshot is a virtual copy of a disk volume. The snapshot appears to be another disk volume, but actually mostly shares the physical data storage of the original volume. Snapshots solve the backup window problem. Once the snapshot has been taken, system operation can continue while the backup is taken of the snapshot. Effectively, the backup window is extended to the interval between successive backups. The process of creating a virtual volume and taking snapshots is known in the relevant art. See, for example, the method taught by Ohran '953 which functions to retain two snapshots at a time. In the present method, multiple snapshots may be retained, as is discussed in greater detail below.

[0026] Succedent Physical Incremental Backup

[0027] Operation of the processing means **50** can best be explained first with reference to **FIG. 3**, in which the data volume is represented as a sequence of data volume consistent states occurring along a timeline **99**. A data volume base state **101** is defined as the state of the data volume at a baseline time t_0. A base state snapshot **111** (S_0) is made of the data volume base state **101**. A first data volume consistent state **103** occurs at a time $t_1>t_0$, when the data volume is in a subsequent consistent state. The determination of a consistent state can be made, for example, by a network administrator or by an expert system.

[0028] A first state snapshot **113** (S_1) is made of the first data volume consistent state **103**. Similarly, a second data volume consistent state **105** occurs at a time $t_2>t_j$, a third data volume consistent state **107** occurs at a time $t_3>t_2$, and a fourth data volume consistent state **109** occurs at a time $t_4>t_3$ where it is determined that the data volume is in a respective consistent state at times t_2, t_3, and t_4, and so on. A second state snapshot **115** (S_2), a third state snapshot **117** (S_3), and a fourth state snapshot **119** (S_4), corresponding to data volume states **105**, **107**, and **109** respectively, are generated.

[0029] In **FIG. 4**, a full base state backup **130** (B_0) is made of the base state snapshot **111** by copying the entire contents of the base state snapshot **111**. A first succedent snapshot difference list **121** (S_{01}) in data volume state snapshots is then obtained. A 'snapshot difference list' (e.g., $S_0 \rightarrow S_1$) is a list of identifiers of those data blocks in the first state snapshot **113** (S_1) that differ from the data blocks in the base state snapshot **111** (S_0). A 'data block' is a subset of the data volume, typically about 65 K bytes, and is determined according to a user's requirements. As can be appreciated by one skilled in the art, a larger data block size will result in the need to copy larger units of data to snapshot storage. On the other hand, while a smaller data block size will allow for smaller data units to be copied to snapshot storage, this is achieved at the cost of allocating a larger snapshot map to identify the larger number of the smaller data blocks.

[0030] This difference list is then used to list the data blocks that are copied from the snapshot itself to the backup. The first succedent snapshot difference list **121** is generated by identifying those data blocks of the first state snapshot **113** that differ from the data blocks of the base state snapshot **111**. These segments can be identified by examining the snapshot mapping data. The first succedent snapshot difference list **121** thus includes identifiers of all the data blocks of the first state snapshot **113** differing from data blocks in the base snapshot **111**. A first succedent backup **131** (B_{01}) is created by copying from the first state snapshot **113** (S_1) all the data blocks identified in the first succedent snapshot difference list **121**. A copy of the snapshot difference list **121** is also included in the first succedent backup **131**.

[0031] Once the first succedent backup **131** has been created, the first state snapshot **113** may be partially deleted,

as indicated by dashed lines in **FIG. 5**, and the base state snapshot **111** may be fully deleted. By 'partial deletion' is meant that the mapping metadata for first state snapshot **113** is not deleted. With the deletion of the base state snapshot **111** and the partial deletion of the first state snapshot **113**, it is possible to release blocks in the storage pool containing data unique to the base state snapshot **111** and the first state snapshot **113**. In general, snapshots may be retained online for fast file recovery and/or deleted at a later time.

[0032] As best seen in **FIG. 6**, a second succedent snapshot difference list **123** (S_{12}) in state snapshots (i.e., $S_1 \rightarrow S_2$) is generated. A second succedent backup **133** (B_{12}) is created from the second succedent snapshot difference list **123** by examining the snap disk metadata and copying from the second state snapshot **115** all the data blocks listed in the second succedent snapshot difference list **123** and copying the second succedent snapshot difference list **123** itself. Once the second succedent backup **133** has been made, the second state snapshot **115** may be partially deleted, leaving at least the mapping metadata for the second state snapshot **115** (S_2), and the first state snapshot **113** may be fully deleted.

[0033] This process is continued: the third state snapshot **117** (S_3) is made of a subsequent consistent state of the data volume; a third succedent snapshot difference list **125** (S_{23}) in state snapshots (i.e., $S_2 \rightarrow S_3$) is generated; and a third succedent backup **135** (B_{23}) is made by copying from the third state snapshot **117** all the data blocks listed in the third succedent snapshot difference list **125** and copying the third succedent snapshot difference list **125** itself.

[0034] Offline Consolidation

[0035] The full base state backup **130**, the first succedent backup **131**, and the second succedent backup **133** are preferably stored offline in conventional memory, such as magnetic and optical media. The full base state backup **130** (B_0) may be consolidated with the first succedent backup **131** (B_0) to yield a new base state backup (i.e., B_1). That is, the operation is equivalent to making a full base state backup of the first state snapshot **113** (S_1). This is done by selectively copying the contents of the full base state backup **130** (B_0) and copying the contents of the first succedent backup **131** (B_{01}) such that the new base state backup (B_1) includes: i) the entire contents of the first succedent backup **131** (B_{01}), and ii) those blocks in the full base state backup **130** (B_0) which are not present in the first succedent backup **131** (B_{01}). This operation may be performed in an offline manner, that is, without making reference to the online data.

[0036] In comparison to the consolidation of conventional file-structured backups, consolidation of logical volumes as disclosed herein is more direct. A logical volume is a set of data blocks, where each data block may be read and written only. Thus, consolidating physical incremental save sets is relatively simple with the results being assured. If desired, a verification function can be included to verify the correctness of saved backups. For example, a consolidated backup can be created by consolidating the full base state backup **130** (B_0), the first succedent backup **131** (B_{01}), and the second succedent backup **133** (i.e., B_{12}). It can then be verified that the resulting consolidated backup is equivalent to a full second state backup B_2 (not shown) by comparing its contents with the contents of the second state snapshot **115** while the second state is snapshot **115** is still online.

[0037] One advantage of offline consolidation is that the consolidation process requires no bandwidth in the primary data store at the expense of requiring additional equipment in the form of tape drives and processing power. An important characteristic of the physical incremental backup is that only those data blocks that have changed are copied and not entire files. With a file-based incremental backup, changing just one record in a file causes the entire file to be backed up. With a physical incremental, only the data blocks containing the updated record, and possibly affected index blocks, are backed up.

[0038] Incremental Data Volume Restore

[0039] To recover the data volume in the present example, the backups are restored in successive order. The full base state backup **130** (B_0) is obtained and subsequently overwritten with the first succedent backup **131** and then with the second succedent backup **133**. This yields an exact copy of the volume as of its second state snapshot **115**. Alternately, if the full base state backup **130** (B_0) and the subsequent incrementals B_{01} and B_{12} had been previously consolidated into a single backup, a simple restoring procedure would also restore the volume to its state at second state snapshot **115** (S_2).

[0040] The process of succedent incremental backups works well for disaster recovery (i.e., situations in which the entire volume has been lost), but is not practical for the recovery of individual files. Ideally, the individual file is recovered by copying it from a snapshot that is still online. However, if snapshots have been rolled out and deleted, restoring a snapshot requires, in effect, recovering the entire volume by restoring the base state snapshot **111** and all subsequent snapshots up to the one containing the desired file. The succedent incremental backup process for recovery of a file is thus equivalent to a full physical restore of the data volume.

[0041] Precedent Physical Incremental Backup

[0042] File recovery is best accomplished when a precedent physical incremental backup has been performed. This process is illustrated in **FIG. 7**, where a base state snapshot difference list **141** (S_{10}) in state snapshots (i.e., $S_1 \rightarrow S_0$) is generated by identifying all segments of the base state snapshot **111** that are different from the first state snapshot **113**. A base state backup **151** (B_{10}) is made by copying from the first state snapshot **113** all the data blocks identified in the base state snapshot difference list **141**. Once the copying step has been performed, the base state snapshot **111** can be deleted.

[0043] A first precedent snapshot difference list **143** (S_{21}) in state snapshots (i.e., $S_2 \rightarrow S_1$) is generated. All segments of the first state snapshot **113** that are different from the second state snapshot **115** are listed. A first precedent backup **153** (B_{21}) is made by copying from the first state snapshot **113** all the data blocks identified in the first precedent snapshot difference list **143** and by copying the first precedent snapshot difference list **143**. Once the copying steps have been performed, the first state snapshot **113** can be deleted (not shown). The precedent physical incremental backup process is continued to obtain a second precedent snapshot difference list **145** (S_{32}) and a second precedent backup **155** (B_{32}), and a third precedent snapshot difference list **147** (S_{43}) and a third precedent backup **157** (B_{43}) in a similar manner.

[0044] Incremental File Restore

[0045] Restoration of a file resident in an antecedent snapshot can be effectively accomplished by rolling back from a snapshot that is still online. For example, the fourth state snapshot **119**, as shown in **FIG. 8**, is still online at the time t_4. The restoration of the first state snapshot **113** can then be accomplished by means of the following procedure.

[0046] A duplicate fourth state snapshot **119'** (D_4) is cloned from the fourth state snapshot **119** and overwritten with the contents of the third precedent backup **157**. This produces a copy of the third state snapshot **117**. The incremental storage and time required to restore are directly proportional to the amount of change between the two respective snapshots.

[0047] The third state snapshot **117** is then overwritten with the contents of the second precedent backup **155** to give the second state snapshot **115**, as shown in **FIG. 9**. After the second state snapshot **115** is overwritten with the first precedent backup **153**, the first state snapshot **113** is obtained. More generally, any snapshot can be restored using a series of precedent or succedent incrementals by duplicating or cloning the oldest (or nearest) available online snapshot and then restoring the necessary chain of backups. For example, if the base state snapshot **111** were still online, one could recover the first state snapshot **113** by cloning the base state snapshot **111** to produce a duplicate base state snapshot **111'** and then overwriting the duplicate base state snapshot **111'** with the contents of the first succedent backup **131**.

[0048] In an alternative embodiment, a 'conditional overwrite,' rather than a complete overwrite, is performed in the file recovery process. As can be appreciated by one skilled in the relevant art, each complete overwrite results in a copy-out which consumes real storage space. In a conditional overwrite, the existing data blocks listed in the third state snapshot **117**, for example, are compared with the 'new' data blocks listed in the second precedent backup **155**. If a new data block is the same as the respective data block listed in the third state snapshot **117**, then the respective data block listed in the third state snapshot **117** is not written over. If the new data block is not the same as the respective data block listed in the third state snapshot **117**, then the respective data block listed in the third state snapshot **117** is written over. This technique is especially important when restoring composite backups, as described in greater detail below.

[0049] Offline Consolidation of Precedent Backups

[0050] Successive precedent backups may be combined into a single precedent backup to reduce offline storage volume and to speed incremental file recovery, as shown in **FIG. 10**. In way of example, the second precedent backup **155** and the first precedent backup **153** can be combined into a concatenated precedent backup **161** (B_{31}) by copying the contents of the first precedent backup **153** in its entirety, and including only the contents of the second precedent backup **155** where corresponding blocks are not present in the first precedent backup **153**. The first precedent snapshot difference list **153** and the second precedent snapshot difference list **155** are also copied into the concatenated precedent backup **161**.

[0051] By way of further example, the third precedent backup **157** and the second precedent backup **155** are combined into a concatenated precedent backup **163** (B_{42}) by copying the contents of the second precedent backup **155** in its entirety, and including only the contents of the third precedent backup **157** where corresponding blocks are not present in the second precedent backup **155**. The concatenated precedent backup **163** is thus a concatenation of the third precedent backup **157** and the second precedent backup **155**, and should not be considered as simply a precedent backup. In a subsequent operation, the concatenated precedent backup **163** may be combined with the first precedent backup **153** into a concatenated precedent backup **165** (B_{41}) in a similar manner, that is by copying the contents of the first precedent backup **153** in its entirety, and including only the contents of the concatenated precedent backup **163** where corresponding blocks are not present in the first precedent backup **153**. The concatenated precedent backup **165** also includes the all three precedent snapshot difference lists **143**, **145**, and **147**. Thus, the first state snapshot **113** can be directly obtained from the online fourth state snapshot **119** by overwriting the duplicate fourth state snapshot **119'** with the concatenated precedent backup **165**, as shown in **FIG. 11**. A similar procedure can be used to obtain the first state snapshot **113** from the online third state snapshot **117** by overwriting a duplicate third state snapshot **117'** with the concatenated precedent backup **161** (not shown).

[0052] Moreover, as is apparent to one skilled in the relevant art, the concatenated precedent backup **165** can also be used to restore the first state snapshot **113** from a duplicate of the second state snapshot **115** or a duplicate of the third state snapshot **117**, if available. In such a restoration operation, the conditional overwrite process described above is particularly important as the concatenated precedent backup **165** contains blocks whose contents match the contents of the corresponding blocks in the second state snapshot **115** or the third state snapshot **117**, and are therefore redundant for the restoration process.

[0053] In yet another embodiment, unnecessary copy-outs can be avoided by using the is list of differences contained in the respective precedent backup. The first precedent backup **153** contains a first differences list, which can be denoted by L_{21}. Similarly, the second precedent backup **155** contains a second differences list denoted by L_{32}, and the third precedent backup **157** contains a third differences list denoted by L_{43}. When a series of precedent backups are concatenated, there are included copies of all the differences lists of the individual backups incorporated into the respective concatenated backup. For example, the concatenated precedent backup **165**, which is created by combining the third precedent backup **157**, the second precedent backup **155**, and the first precedent backup **153**, includes the first differences list L_{21}, the second differences list L_{32} and the third differences list L_{43}. Thus, if the concatenated precedent backup **165** is used to restore the first state snapshot **113** by using a duplicate second subsequent snapshot D_2 (not shown), only the first differences list L_{21} is needed to select blocks from the concatenated precedent backup **165** for restoration. However, if the concatenated precedent backup **165** is used to restore the first state snapshot **113** by using the duplicate fourth state snapshot **119'**, all three differences lists L_{21}, L_{32}, and L_{43} are needed to select blocks from the concatenated precedent backup **165** for restoration.

[0054] Offline Consolidation of Succedent Backups

[0055] In a method similar to the concatenation of precedent backups, described above, successive succedent backups may be combined into a single succendent backup to reduce offline storage volume and to speed disaster recovery, as shown in **FIG. 12**. The second succedent backup **135**, for example, and a third succedent backup **137** (B_{34}), obtained from a fourth succedent snapshot difference list **127**, can be combined into a concatenated succedent backup **173** (B_{24}) by copying the contents of the third succedent backup **137** in its entirety, and including only the contents of the second succedent backup **135** where corresponding blocks are not present in the third succedent backup **137**. The second succedent snapshot difference list **125** and the third succedent snapshot difference list **127** are also copied into the concatenated succedent backup **173**.

[0056] As explained above, the invention is not limited to the process of concatenating only successive backups but also includes the concatenation of multiple succedent backups. The first succedent backup **133**, for example, and the second succedent backup **135** can be combined into a concatenated succedent backup **171** (B_{13}) by copying the contents of the second succedent backup **135** in its entirety, and including only the contents of the first succedent backup **133** where corresponding blocks are not present in the second succedent backup **135**. The first succedent snapshot difference list **123** and the second succedent snapshot difference list **125** are also copied into the concatenated succedent backup **171**.

[0057] The third succedent backup **137** can then be combined with the concatenated succedent backup **171** into a concatenated succedent backup **175** (B_{14}) by copying the contents of the third succedent backup **137** in its entirety, and including only the contents of the concatenated succedent backup **171** where corresponding blocks are not present in the third succedent backup **137**. The concatenated succedent backup **175** also includes all three succedent snapshot difference lists **123**, **125**, and **127**. The concatenated succedent backup **175** is thus a concatenation of the third succedent backup **137** and the concatenated succedent backup **171**, and should not be considered as simply a succedent backup. The fourth state snapshot **119**, therefore, can be directly obtained from the first state snapshot **113** by overwriting a duplicate first state snapshot **113'** with the concatenated succedent backup **175**, as shown in **FIG. 13**.

[0058] Composite Physical Incremental Backup

[0059] In a preferred embodiment, succedent physical incremental backup and precedent physical incremental backup are combined into a series of composite incremental backups, as shown in **FIGS. 14 and 15**. A first composite backup **181** (C_{012}) includes both the first succedent snapshot difference list **121** and the first precedent snapshot difference list **143**. A second composite backup **183** (C_{123}) includes both the second succedent snapshot difference list **123** and the second precedent snapshot difference list **145**. The first composite backup **181** and the second composite backup **183** are stored offline.

[0060] In general, the composite physical incremental backup procedure begins by first acquiring the base state snapshot **111** (S_0) of a data volume base state at a time t_0. A successive series of n snapshots, S_1 through S_n are then acquired at data volume consistent states occurring at times $t_1<t_2< \ldots <t_n$ respectively. As the snapshots are acquired, three sets of derived products are generated or created. For the first derived set, a j^{th} succedent snapshot difference list $S_{(j-1)(j)}$ is generated as each corresponding snapshot S_j is acquired, where $1 \leqq j \leqq n$. In all, a series of n succedent snapshot difference lists S_{01} through $S_{(n-1)(n)}$ is obtained from the succession of n snapshots S_1 through S_n. For the second derived set, a $(j-1)^{th}$ precedent snapshot difference list $S_{(j)(j-1)}$ is generated as each corresponding snapshot S_j is acquired. In all, a series of n precedent snapshot difference lists S_{10} through $S_{(n)(n-1)}$ is obtained from the succession of n snapshots S_1 through S_n.

[0061] An initial base state backup **130** (B_0) is made by copying the contents of the initial base state snapshot (S_0) in its entirety. The initial precedent incremental base state backup **151** (B_{10}) is made by copying from the base state snapshot **111** (S_0) all the blocks in the base state snapshot difference list **141** (S_{10}). For the third derived set, a $(j-1)^{th}$ composite backup $C_{(j-2)(j-1)(j)}$ is created by copying from the $(j-1)^{th}$ state snapshot all the data blocks identified in the corresponding $(j-1)^{th}$ succedent snapshot difference list $S_{(j-2)(j-1)}$, and by copying from the $(j-1)^{th}$ state snapshot all the data blocks identified in the $(j-1)^{th}$ precedent snapshot difference list $S_{(j)(j-1)}$ to produce a series of (n−1) composite backups C_{012} through $C_{(n-2)(n-1)(n)}$. Preferably, each snapshot S_1 through $S_{(n-1)}$ is tagged with a unique identifier and each composite backup C_{012} through $C_{(n-2)(n-1)(n)}$ is tagged with the unique identifiers corresponding to the snapshots from which it is derived. Each composite backup $C_{(j-2)(j-1)(j)}$ is thus tagged with the unique identifiers of snapshots $S_{(j-2)}$, $S_{(j-1)}$, and $S_{(j)}$, where snapshots $S_{(j-2)}$ and $S_{(j)}$ are referred to as difference snapshots, and $S_{(j-1)}$ is referred to as the contents snapshot. The unique identifiers may be used in a data recovery procedure.

[0062] In the composite physical incremental backup process, disaster recovery is accomplished by restoring the base state backup **130** (B_0) and overwriting it with the succedent incrementals (i.e., B_{01}, C_{012}, C_{123}, . . .), as described for the succedent physical incremental backup procedure above. File recovery is accomplished by rolling back from an online snapshot (e.g., S_j) by successively overwriting a duplicate (D_j) of the online snapshot with the composite incrementals ($C_{(j-2)(j-1)(j)}$, $C_{(j-3)(j-2)(j-1)}$, $C_{(j-4)(j-3)(j-2)}$, . . .), in a similar manner as described above for the precedent physical incremental backup procedure. At each overwrite step, the current unique identifier of the snapshot being overwritten is compared to the unique identifiers of the difference snapshots in the composite backup. After the overwrite has been completed, the unique identifier of the updated snapshot is set to be the unique identifier of the contents snapshot of the composite backup. This ensures that the correct incremental is being used.

[0063] While the invention has been described with reference to particular embodiments, it will be understood that the present invention is by no means limited to the particular constructions and methods herein disclosed and/or shown in the drawings, but also comprises any modifications or equivalents within the scope of the claims.

What is claimed is:

1. A method of protecting computer data, said method comprising the steps of:

acquiring a first state snapshot S_1 of a first data volume consistent state at a time t_1;

acquiring a second state snapshot S_2 of a second data volume consistent state at a time $t_2>t_1$;

generating a first precedent snapshot difference list S_{21} comprising an identification of data blocks of said first state snapshot S_1 differing from data blocks in said second state snapshot S_2; and

creating a first precedent backup B_{21} by copying from said first state snapshot S_1 data blocks identified in said first precedent snapshot difference list S_{21}, said first precedent backup B_{21} further comprising said first precedent snapshot difference list S_{21}.

2. The method of claim 1 further comprising the step of storing said first precedent backup B_{21} in an offline memory means.

3. The method of claim 2 wherein said offline memory means comprises at least one member from the group consisting of magnetic tape and optical disk.

4. The method of claim 1 further comprising the step of deleting said first state snapshot S_1 following said step of generating a first precedent snapshot difference list S_{21}.

5. The method of claim 2 further comprising the steps of:

retrieving said first precedent backup B_{21};

recovering said first precedent snapshot difference list S_{21} from said first precedent backup B_{21}; and

restoring said first state snapshot S_1 by overwriting at least a portion of said second state snapshot S_2 with the contents of said first precedent backup B_{21}.

6. The method of claim 1 further comprising the steps of:

acquiring a third state snapshot S_3 of a third data volume consistent state at a time $t_3>t_2$;

generating a second precedent snapshot difference list S_{32} comprising an identification of data blocks of said second state snapshot S_2 differing from data blocks in said third state snapshot S_3; and

creating a second precedent backup B_{32} by copying from said second state snapshot S_2 data blocks identified in said second precedent snapshot difference list S_{32}, said second precedent backup B_{32} further comprising said second precedent snapshot difference list S_{32}.

7. The method of claim 6 further comprising the step of storing said second precedent backup B_{32} in an offline memory means.

8. The method of claim 6 further comprising the step of deleting said second state snapshot S_2 following said step of generating a second precedent snapshot difference list S_{32}.

9. The method of claim 6 further comprising the steps of:

recovering said second precedent snapshot difference list S_{32} from said second precedent backup B_{32}; and

restoring said second state snapshot S_2 by overwriting at least a portion of said third state snapshot S_3 with the contents of said second precedent backup B_{32}.

10. The method of claim 6 further comprising the steps of:

generating a concatenated precedent snapshot difference list S_{31} comprising an identification of said data blocks of said second state snapshot S_2 differing from data blocks in said third state snapshot S_3 and an identification of said data blocks of said first state snapshot S_1 differing from data blocks in said second state snapshot S_2;

creating a concatenated backup B_{31} by copying all said blocks in said first precedent backup B_{21} and copying all blocks in said second precedent backup B_{32} not present in said first precedent backup B_{21};

copying said first precedent snapshot difference list S_{21} and said second precedent snapshot difference list S_{32} into said concatenated backup B_{31}; and

storing said concatenated backup B_{31} in an offline memory means.

11. The method of claim 10 further comprising the steps of:

retrieving said concatenated backup B_{31};

recovering said concatenated precedent snapshot difference list S_{31} from said concatenated backup B_{31}; and

restoring said first state snapshot S_1 by overwriting at least a portion of said third state snapshot S_3 with the contents of said concatenated backup B_{31}.

12. The method of claim 10 further comprising the steps of:

acquiring a fourth state snapshot S_4 of a fourth data volume consistent state at a time $t_4>t_3$;

generating a third precedent snapshot difference list S_{43} comprising an identification of data blocks of said third state snapshot S_3 differing from data blocks in said fourth state snapshot S_4;

creating a third precedent backup B_{43} by copying from said third state snapshot S_3 data blocks identified in said third precedent snapshot difference list S_{43}, said third precedent backup B_{43} further comprising said third precedent snapshot difference list S_{43},

generating a concatenated precedent snapshot difference list S_{42} comprising an identification of said data blocks of said third state snapshot S_3 differing from data blocks in said fourth state snapshot S_4 and an identification of said data blocks of said second state snapshot S_2 differing from data blocks in said third state snapshot S_3;

creating a concatenated backup B_{42} by copying all said blocks in said second precedent backup B_{32} and copying all blocks in said third precedent backup B_{43} not present in said second precedent backup B_{32}, and copying said second and third precedent difference lists S_{32} and S_{43} into said concatenated backup B_{42}; and

creating a concatenated backup B_{41} by copying all said blocks in said first precedent backup B_{21} and copying all blocks in said concatenated backup B_{42} not present in said first precedent backup B_{21}, and copying first, second, and third precedent difference lists S_{21}, S_{32} and S_{43}, into said concatenated backup B_{41}.

13. The method of claim 12 further comprising the steps of:

retrieving said concatenated backup B_{41};

recovering said concatenated precedent snapshot difference list S_{41} from said concatenated backup B_{41}; and

restoring said first state snapshot S_1 by overwriting at least a portion of said fourth state snapshot S_4 with the contents of said concatenated backup B_{41}.

14. A method of protecting computer data, said method comprising the steps of:

acquiring a base state snapshot S_0 of a data volume base state at a time t_0;

creating a base state backup B_0 of said base state snapshot S_0;

acquiring a first state snapshot S_1 of a first data volume consistent state at a time $t_1>t_0$;

generating a first succedent snapshot difference list S_{01} comprising an identification of data blocks of said first state snapshot S_1 differing from data blocks in said base state snapshot S_0;

creating a first succedent backup B_{01} by copying from said first state snapshot S_1 data blocks identified in said first succedent snapshot difference list S_{01}, and copying said first succedent snapshot difference list S_{01}; and

deleting at least a portion of said first state snapshot S_1.

15. The method of claim 14 further comprising the step of storing said base state backup B_0 and said first succedent backup B_0 in an offline memory means.

16. The method of claim 15 further comprising the steps of:

retrieving said base state backup B_0 and said first succedent backup B_{01};

recovering said base state snapshot S_0 and said first succedent snapshot difference list S_{01} from said base state backup B_0 and said first succedent backup B_{01}, respectively; and

restoring said first state snapshot S_1 by overwriting said base state snapshot S_0 with said first succedent backup B_{01}.

17. The method of claim 14 further comprising the steps of:

acquiring a second state snapshot S_2 of a second data volume consistent state at a time $t_2>t_1$;

generating a second succedent snapshot difference list S_{12} comprising an identification of data blocks of said second state snapshot S_2 differing from data blocks in said first state snapshot S_1;

creating a second succedent backup B_{12} by copying from said second state snapshot S_2 data blocks identified in said second succedent snapshot difference list S_{12}, and copying said second succedent snapshot difference list S_{12}; and

deleting at least a portion of said second state snapshot S_2.

18. A method of protecting computer data, said method comprising the steps of:

acquiring a base state snapshot S_0 of a data volume base state at a time to;

acquiring a first state snapshot S_1 of a first data volume consistent state at a time $t_1>t_0$;

acquiring a second state snapshot S_2 of a second data volume consistent state at a time $t_2>t_1$;

generating a first succedent snapshot difference list S_{01} comprising an identification of data blocks of said first state snapshot S_0 differing from data blocks in said base state snapshot S_0;

generating a first precedent snapshot difference list S_{21} comprising an identification of data blocks of said first state snapshot S_1 differing from data blocks in said second state snapshot S_2;

creating a first composite backup C_{012} by copying from said first state snapshot S_1 data blocks identified in first succedent snapshot difference list S_{01} and copying from said first state snapshot S_1 data blocks identified in said first precedent snapshot difference list S_{21} and

copying said first succedent snapshot difference list S_{01} and said first precedent snapshot difference list S_{21} into said first composite backup C_{012}.

19. The method of claim 18 further comprising the step of storing said first composite backup C_{012} in an offline memory means.

20. The method of claim 19 further comprising the steps of:

retrieving said first composite backup C_{012};

recovering said first precedent snapshot difference list S_{21}; and

restoring said first state snapshot S_1 by overwriting at least a portion of said second state snapshot S_2 with at least a portion of the contents of said first composite backup C_{012}.

21. The method of claim 18 further comprising the steps of:

acquiring a third state snapshot S_3 of a third data volume consistent state at a time $t_3>t_2$;

generating a second succedent snapshot difference list S_{12} comprising an identification of data blocks of said second state snapshot S_2 differing from data blocks in said first state snapshot S_1;

generating a second precedent snapshot difference list S_{32} comprising an identification of data blocks of said second state snapshot S_2 differing from data blocks in said third state snapshot S_3;

creating a second composite backup C_{123} by copying from said second state snapshot S_2 data blocks identified in second succedent snapshot difference list S_{12} and copying from said second state snapshot S_2 data blocks identified in said second precedent snapshot difference list S_{32} and

copying said second succedent snapshot difference list S_{12} and said second precedent snapshot difference list S_{32} into said second composite backup C_{123}.

22. The method of claim 21 further comprising the step of storing said second composite backup C_{123} in an offline memory means.

23. The method of claim 18 further comprising the steps of:

acquiring a plurality of third through n^{th} state snapshots S_3 through S_n, of third through n^{th} data volume consistent states at respective times $t_3 < \ldots t_j \ldots \leqq t_n$;

generating second through $(n-1)^{th}$ succedent snapshot difference lists S_{12} through $S_{(n-2)(n-1)}$ respectively, a $(j-1)^{th}$ succedent snapshot difference list $S_{(j-2)(j-1)}$ comprising an identification of data blocks of a $(j-1)^{th}$ state snapshot $S_{(j-1)}$ differing from data blocks in a $(j-2)^{th}$ state snapshot $S_{(j-2)}$;

generating second through $(n-1)^{th}$ precedent snapshot difference lists S_{32} through $S_{(n)(n-1)}$ respectively, a $(j-1)^{th}$ precedent snapshot difference list $S_{(j)(j-1)}$ comprising an identification of data blocks of a $(j-1)^{th}$ state snapshot $S_{(j-1)}$ differing from data blocks in a j^{th} state snapshot S_j;

creating second through $(n-1)^{th}$ composite backups C_{123} through $C_{(n-2)(n-1)(n)}$, wherein a $(j-1)^{th}$ composite backup $C_{(j-2)(j-1)(j)}$ is created by copying from said $(j-1)^{th}$ state snapshot $S_{(j-1)}$ data blocks identified in said $(j-1)^{th}$ succedent snapshot difference list $S_{(j-2)(j-1)}$ and copying from said $(j-1)^{th}$ state snapshot $S_{(j-1)}$ data blocks identified in said $(j-1)^{th}$ precedent snapshot difference list $S_{(j)(j-1)}$ and

copying said succedent snapshot difference lists S_{12} through $S_{(n-2)(n-1)}$ and said precedent snapshot difference lists S_{32} through $S_{(n)(n-1)}$ into said respective composite backups C_{123} through $C_{(n-2)(n-1)(n)}$.

24. The method of claim 23 further comprising the step of storing one or more of said second through $(n-1)^{th}$ composite backups C_{123} through $C_{(n-2)(n-1)(n)}$ in an offline memory means.

25. The method of claim 23 further comprising the step of deleting said third through $(n-1)^{th}$ state snapshots S_3 through S_{n-1}.

26. The method of claim 23 further comprising the steps of:

assigning a unique identifier to each said state snapshot S_j for each said composite backup $C_{(j-2)(j-1)(j)}$, identifying said state snapshots $S_{(j-2)}$ and S_j as difference snapshots and said $(j-1)^{th}$ state snapshot $S_{(j-1)}$ as a content snapshot;

for each said composite backup $C_{(j-2)(j-1)(j)}$, copying the unique identifiers of said state snapshots $S_{(j-2)}$, $S_{(j-1)}$, and S_j into said $(j-1)^{th}$ composite backup $C_{(j-2)(j-1)(j)}$.

27. The method of claim 26 further comprising the steps of:

retrieving said $(j-1)^{th}$ composite backup $C_{(j-2)(j-1)(j)}$;

recovering said $(j-1)^{th}$ precedent snapshot difference list $S_{(j)(j-1)}$ from said $(j-1)^{th}$ composite backup $C_{(j-2)(j-1)(j)}$; and

restoring said $(j-1)^{th}$ state snapshot S_{j-1} by overwriting at least a portion of said j^{th} state snapshot S_j with the contents of said $(j-1)^{th}$ composite backup $C_{(j-2)(j-1)(j)}$.

28. The method of claim 26 further comprising the step of comparing the unique identifier of said j^{th} snapshot S_j to the unique identifiers of the difference snapshots of said $(j-1)^{th}$ composite backup $C_{(j-2)(j-1)(j)}$.

29. The method of claim 26 further comprising the step of assigning the unique identifier of the content snapshot of said $(j-1)^{th}$ composite backup $C_{(j-2)(j-1)(j)}$ to be the unique identifier of said restored $(j-1)^{th}$ state snapshot $S_{(j-1)}$.

30. The method of claim 23 further comprising the steps of:

creating a $(j-1)^{th}$ precedent backup $B_{(j-1)(j-2)}$ by copying from said $(j-1)^{th}$ state snapshot $S_{(j-1)}$ data blocks identified in said $(j-1)^{th}$ precedent snapshot difference list $S_{(j)(j-1)}$, said $(j-1)^{th}$ precedent backup $B_{(j)(j-1)}$ further comprising said $(j-1)^{th}$ precedent snapshot difference list $S_{(j)(j-1)}$;

creating a $(j-2)^{th}$ precedent backup $B_{(j-1)(j-2)}$ by copying from said $(j-2)^{th}$ state snapshot $S_{(j-2)}$ data blocks identified in said $(j-2)^{th}$ precedent snapshot difference list $S_{(j-1)(j-2)}$, said $(j-2)^{th}$ precedent backup $B_{(j-1)(j-2)}$ further comprising said $(j-2)^{th}$ precedent snapshot difference list $S_{(j-1)(j-2)}$; and

creating a first concatenated precedent backup $B_{(j)(j-2)}$ from said $(j-1)^{th}$ precedent backup $B_{(j)(j-1)}$ and said $(j-2)^{th}$ precedent backup $B_{(j-1)(j-2)}$ by copying all blocks in said $(j-2)^{th}$ precedent backup $B_{(j-1)(j-2)}$ and by also copying all blocks in said $(j-1)^{th}$ precedent backup $B_{(j)(j-1)}$ not present in said $(j-2)^{th}$ precedent backup $B_{(j-1)(j-2)}$ and by copying said precedent difference lists $S_{(j-1)(j-2)}$ and $S_{(j)(j-1)}$ from said precedent backups $B_{(j-1)(j-2)}$ and $B_{(j)(j-1)}$ into said concatenated precedent backup $B_{(j)(j-2)}$.

31. The method of claim 30 further comprising the steps of:

creating a $(j-3)^{th}$ precedent backup $B_{(j-2)(j-3)}$ by copying from said $(j-3)^{th}$ state snapshot $S_{(j-3)}$ data blocks identified in said $(j-3)^{th}$ precedent snapshot difference list $S_{(j-2)(j-3)}$, said $(j-3)^{th}$ precedent backup $B_{(j-2)(j-3)}$ further comprising said $(j-3)^{th}$ precedent snapshot difference list $S_{(j-2)(j-3)}$; and

creating a second concatenated precedent backup $B_{(j)(j-3)}$ from said first concatenated precedent backup $B_{(j)(j-2)}$ and said $(j-3)^{th}$ precedent backup $B_{(j-2)(j-3)}$ by copying all blocks in said $(j-3)^{th}$ precedent backup $B_{(j-2)(j-3)}$ and by also copying all blocks in said first concatenated precedent backup $B_{(j)(j-2)}$ not present in said $(j-3)^{th}$ precedent backup $B_{(j-2)(j-3)}$, and by copying said precedent difference lists $S_{(j-2)(j-3)}$, $S_{(j-1)(j-2)}$, and $S_{(j)(j-1)}$ from said precedent backup $B_{(j-2)(j-3)}$ and said concatenated precedent backup $B_{(j)(j-2)}$ into said concatenated precedent backup $B_{(j)(j-3)}$.

32. The method of claim 31 further comprising the steps of:

retrieving a concatenated precedent backup $B_{(h)(g)}$, where $g<h \leqq n$; and

restoring a g^{th} state snapshot S_g by overwriting an h^{th} state snapshot S_h with said concatenated precedent backup $B_{(h)(g)}$.

33. The method of claim 23 further comprising the steps of:

creating a $(j-1)^{th}$ succedent backup $B_{(j-2)(j-1)}$ by copying from said $(j-1)^{th}$ state snapshot $S_{(j-1)}$ data blocks identified in said $(j-1)^{th}$ succedent snapshot difference list $S_{(j-2)(j-1)}$, said $(j-1)^{th}$ succedent backup $B_{(j-2)(j-1)}$ further comprising said $(j-1)^{th}$ succedent snapshot difference list $S_{(j-2)(j-)}$;

creating a j^{th} succedent backup $B_{(j-1)(j)}$ by copying from said j^{th} state snapshot S_j data blocks identified in said j^{th} succedent snapshot difference list $S_{(j-1)(j)}$, said j^{th} succedent backup j^{th} further comprising said j^{th} succedent snapshot difference list $S_{(j-1)(j)}$; and

creating a first concatenated succedent backup $B_{(j-2)(j)}$ from said j^{th} succedent backup $B_{(j-1)(j)}$ and said $(j-1)^{th}$ succedent backup $B_{(j-2)(j-1)}$ by copying all blocks in said j^{th} succedent backup $B_{(j-1)(j)}$, and by also copying all blocks in said $(j-1)^{th}$ succedent backup $B_{(j-2)(j-1)}$ not present in said j^{th} succedent backup $B_{(j-1)(j)}$, and by copying said difference lists $S_{(j-1)(j)}$ and $S_{(j-2)(j-1)}$.

34. The method of claim 33 further comprising the steps of:

creating a $(j-2)^{th}$ succedent backup $B_{(j-3)(j-2)}$ by copying from said $(j-2)^{th}$ state snapshot $S_{(j-2)}$ data blocks identified in said $(j-2)^{th}$ succedent snapshot difference list $S_{(j-3)(j-2)}$, said $(j-2)^{th}$ succedent backup $B_{(j-3)(j-2)}$ further comprising said $(j-2)^{th}$ succedent snapshot difference list $S_{(j-3)(j-2)}$; and

creating a second concatenated succedent backup $B_{(j-3)(j)}$ from said first concatenated succedent backup $B_{(j-2)(j)}$ and said $(j-2)^{th}$ succedent backup $B_{(j-3)(j-2)}$ by copying all blocks in said first concatenated succedent backup $B_{(j-2)(j)}$ and by also copying all blocks in said $(j-2)^{th}$ succedent backup $B_{(j-3)(j-2)}$ not present in said first concatenated succedent backup $B_{(j-2)(j)}$, and copying said difference lists $S_{(j-1)(j)}$, $S_{(j-2)(j-1)}$, and $S_{(j-3)(j-2)}$ from said first concatenated succedent backup $B_{(j-2)(j)}$ and said $(j-2)^{th}$ succedent backup $B_{(j-3)(j-2)}$.

35. An apparatus suitable for protecting the data volume in a computer system, said apparatus comprising:

means for acquiring a sequence of state snapshots $S_0, \ldots, S_j, \ldots, S_n$ of the data volume, each said state snapshot acquired at a respective time $t_0 < \ldots t_j \ldots < t_n$;

means for generating a $(j-1)^{th}$ precedent snapshot difference list $S_{(j)(j-1)}$ comprising a list of one or more data blocks of said j^{th} state snapshot S_j identified as differing from data blocks of said $(j-1)^{th}$ state snapshot $S_{(j-1)}$;

means for copying from said $(j-1)^{th}$ state snapshot S_{j-1} all the data blocks listed in said $(j-1)^{th}$ precedent snapshot difference list $S_{(j)(j-1)}$ into a $(j-1)^{th}$ precedent backup $B_{(j)(j-1)}$ and

means for copying said precedent snapshot difference list $S_{(j)(j-1)}$.

36. The apparatus of claim 35 further comprising means for storing said $(j-1)^{th}$ precedent backup $B_{(j)(j-1)}$ in an offline memory means.

37. The apparatus of claim 36 further comprising:

means for retrieving said $(j-1)^{th}$ precedent backup $B_{(j)(j-1)}$;

means for recovering said $(j-1)^{th}$ precedent snapshot difference list $S_{(j)(j-1)}$ from said $(j-1)^{th}$ precedent backup $B_{(j)(j-1)}$; and

means for overwriting at least a portion of said j^{th} state snapshot S_j with at least a portion of the contents of said $(j-1)^{th}$ precedent backup $B_{(j)(j-1)}$.

38. The apparatus of claim 37 further comprising:

means for generating a $(j-1)^{th}$ succedent snapshot difference list $S_{(j-2)(j-1)}$ comprising a list of one or more data blocks of said $(j-1)^{th}$ state snapshot $S_{(j-1)}$ identified as differing from data blocks of said $(j-2)^{th}$ state snapshot $S_{(j-2)}$; and

means for copying from said $(j-1)^{th}$ state snapshot $S_{(j-1)}$ all the data blocks listed in said $(j-1)^{th}$ succedent snapshot difference list $S_{(j-2)(j-1)}$ into said $(j-1)^{th}$ precedent backup $B_{(j)(j-1)}$; and

means for copying said $(j-1)^{th}$ succedent snapshot difference list $S_{(j-2)(j-1)}$.

39. The apparatus of claim 38 further comprising:

means for recovering said $(j-1)^{th}$ succedent snapshot difference list $S_{(j-2)(j-1)}$ from said $(j-1)^{th}$ precedent backup $B_{(j)(j-1)}$; and

means for overwriting at least a portion of said $(j-2)^{th}$ state snapshot $S_{(j-2)}$with at least a portion of the contents of said $(j-1)^{th}$ precedent backup $B_{(j)(j-1)}$.

40. The apparatus of claim 35 further comprising:

means for concatenating an $(h-1)^{th}$ precedent backup $B_{(h)(h-1)}$ with an $(h-2)^{th}$ precedent backup $B_{(h-1)(h-2)}$ through a g^{th} precedent backup $B_{(g+1)(g)}$, where g<h, into a concatenated precedent backup $B_{(h)(g)}$; and

means for storing said concatenated precedent backup $B_{(h)(g)}$ in an offline memory means.

41. The apparatus of claim 38 further comprising:

means for concatenating a $(g+1)^{th}$ succedent backup $B_{(g)(g+1)}$ with a $(g+2)^{th}$ succedent backup $B_{(g+1)(g+2)}$ through an h^{th} succedent backup $B_{(h-1)(h)}$, where g<h, into a concatenated succedent backup $B_{(g)(h)}$; and

means for storing said concatenated succedent backup $B_{(g)(h)}$ in an offline memory means.

* * * * *

US 20020112134A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2002/0112134 A1

Ohran et al. (43) Pub. Date: Aug. 15, 2002

(54) INCREMENTALLY RESTORING A MASS STORAGE DEVICE TO A PRIOR STATE

(76) Inventors: Richard S. Ohran, Henderson, NV (US); Michael R. Ohran, Orem, UT (US)

Correspondence Address:
WORKMAN NYDEGGER & SEELEY
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UT 84111 (US)

(21) Appl. No.: 10/029,189

(22) Filed: Dec. 20, 2001

Related U.S. Application Data

(60) Provisional application No. 60/257,499, filed on Dec. 21, 2000.

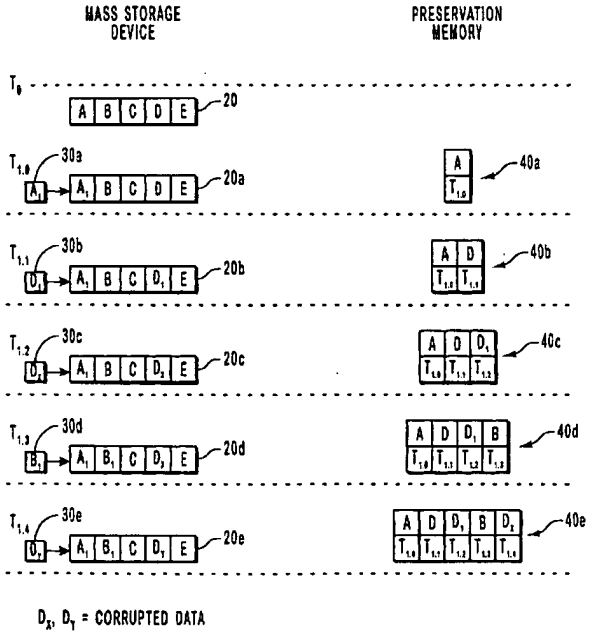
Publication Classification

(51) Int. Cl.[7] G06F 12/16

(52) U.S. Cl. 711/162; 711/114; 714/6

(57) ABSTRACT

Restoring a mass storage device, including the corresponding data blocks stored thereon, to a state in which it existed at a prior instant in time to minimize the data loss caused by data blocks becoming corrupt or lost. After a mirrored or backup copy has been made, data blocks that are to be overwritten in response to a write request are stored in a preservation memory prior to being overwritten. The data blocks stored in the preservation memory are time-stamped to designate the chronological order by which the data blocks were overwritten. If data becomes corrupted, the data blocks of the preservation memory are applied to the corrupted data in reverse chronological order until such time that a valid, non-corrupted set of data is obtained. In this manner, data more recent than that associated with the full mirrored or backup copy can be reconstructed.



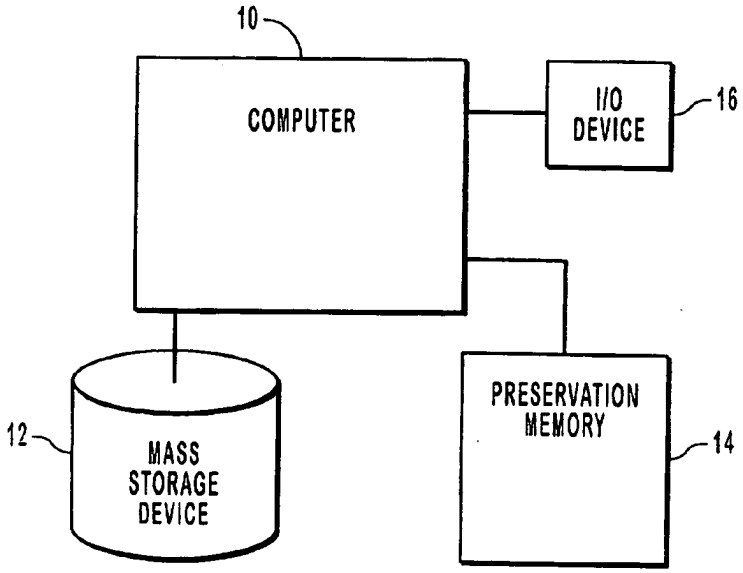

10
COMPUTER
I/O DEVICE
16
12
MASS STORAGE DEVICE
PRESERVATION MEMORY
14

FIG. 1

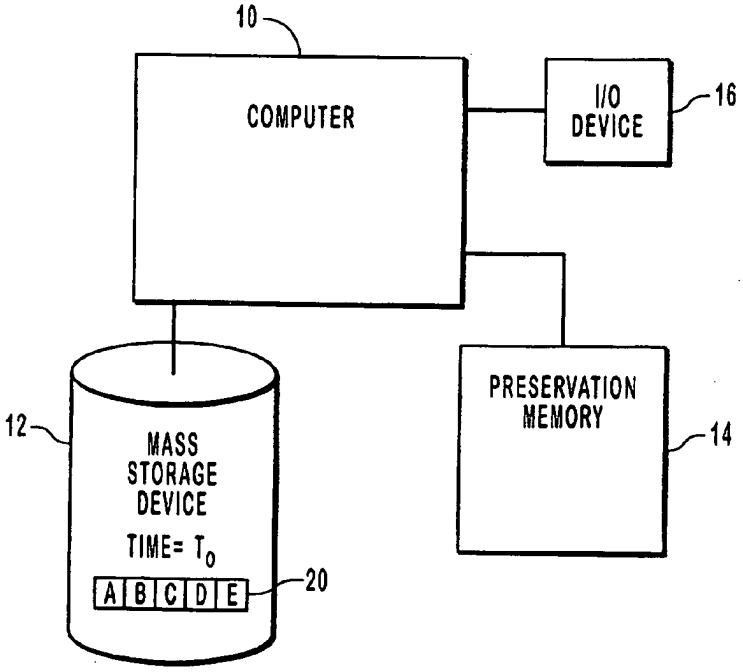

10
COMPUTER
I/O DEVICE
16
12
MASS STORAGE DEVICE
TIME= T_0
A B C D E
20
PRESERVATION MEMORY
14

FIG. 2

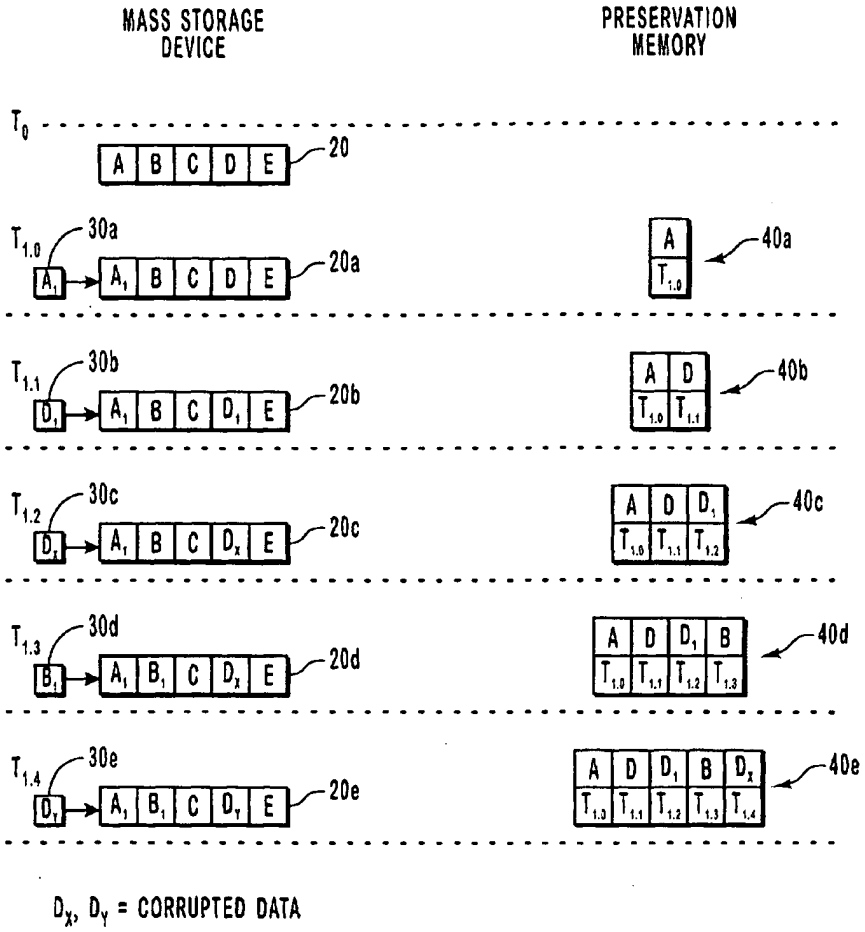

MASS STORAGE DEVICE
PRESERVATION MEMORY
T_0
A B C D E
20
$T_{1.0}$
30a
A_1
A_1 B C D E
20a
A
$T_{1.0}$
40a
$T_{1.1}$
30b
D_1
A_1 B C D_1 E
20b
A D
$T_{1.0}$ $T_{1.1}$
40b
$T_{1.2}$
30c
D_X
A_1 B C D_X E
20c
A D D_1
$T_{1.0}$ $T_{1.1}$ $T_{1.2}$
40c
$T_{1.3}$
30d
B_1
A_1 B_1 C D_X E
20d
A D D_1 B
$T_{1.0}$ $T_{1.1}$ $T_{1.2}$ $T_{1.3}$
40d
$T_{1.4}$
30e
D_Y
A_1 B_1 C D_Y E
20e
A D D_1 B D_X
$T_{1.3}$ $T_{1.1}$ $T_{1.2}$ $T_{1.3}$ $T_{1.4}$
40e
D_X, D_Y = CORRUPTED DATA

FIG. 3

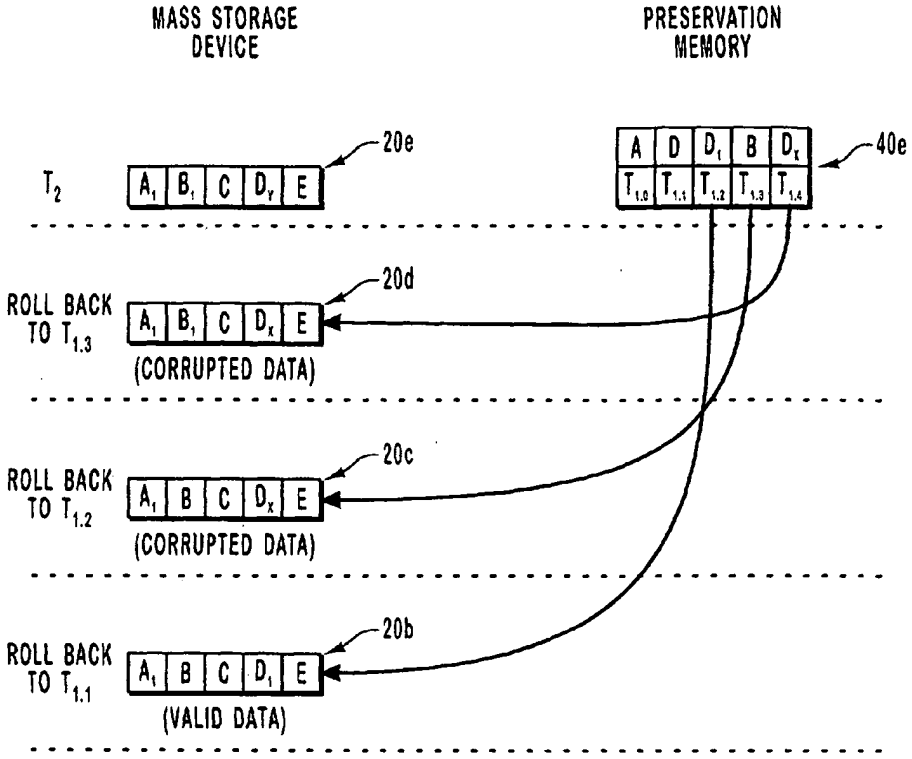

MASS STORAGE DEVICE
PRESERVATION MEMORY
20e
40e
T_2
A_1 B_1 C D_x E
A D D_1 B D_x
$T_{1.0}$ $T_{1.1}$ $T_{1.2}$ $T_{1.3}$ $T_{1.4}$
20d
ROLL BACK TO $T_{1.3}$
A_1 B_1 C D_x E
(CORRUPTED DATA)
20c
ROLL BACK TO $T_{1.2}$
A_1 B C D_x E
(CORRUPTED DATA)
20b
ROLL BACK TO $T_{1.1}$
A_1 B C D_1 E
(VALID DATA)

FIG. 4

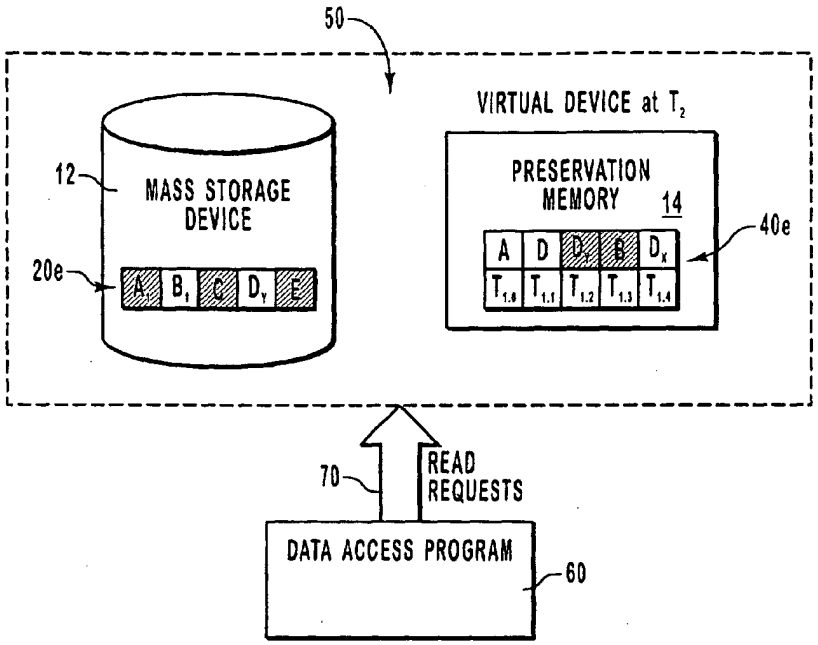
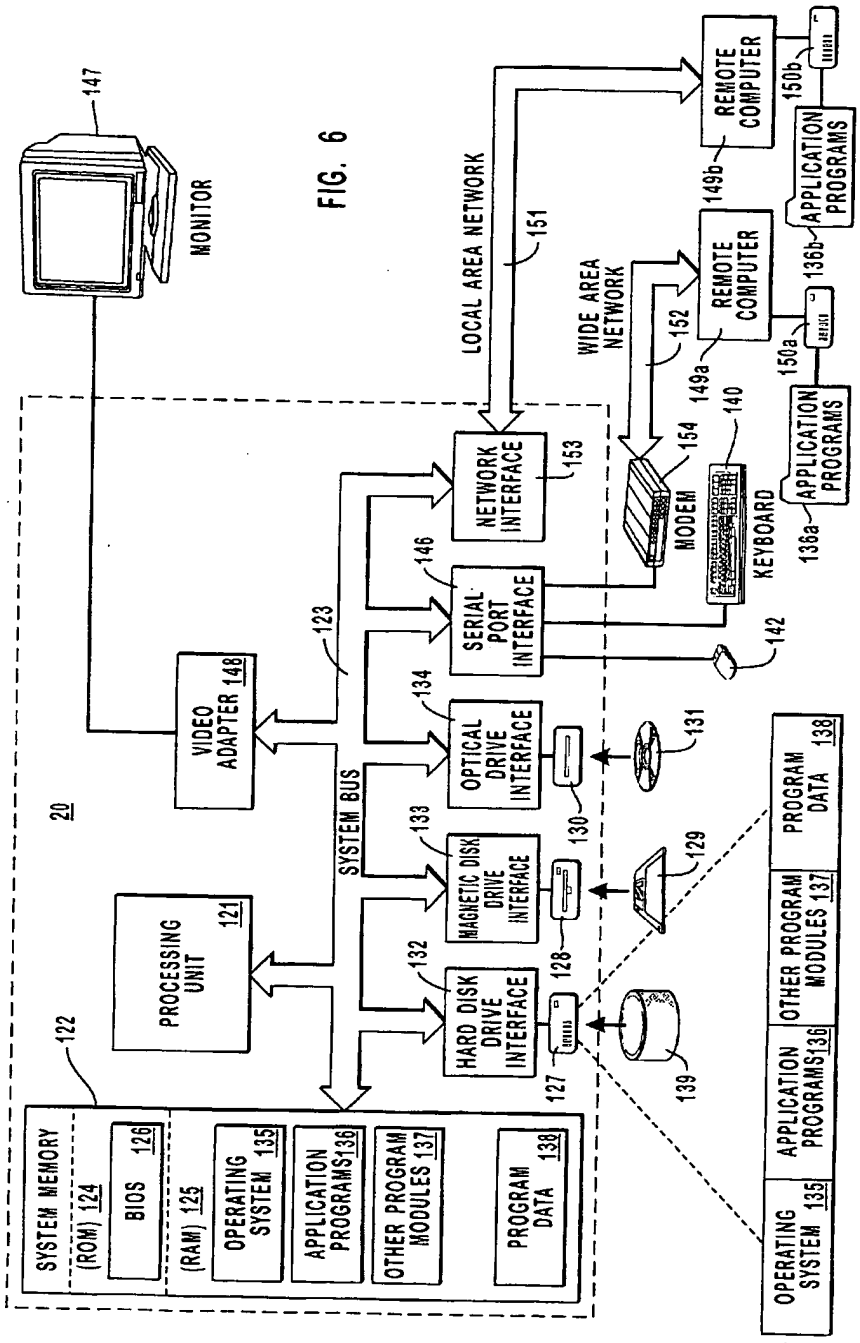

50
VIRTUAL DEVICE at T_2
12
MASS STORAGE DEVICE
20e
A_1 B_1 C D_Y E
PRESERVATION MEMORY 14
40e
A D D_1 B D_X
$T_{1.0}$ $T_{1.1}$ $T_{1.2}$ $T_{1.3}$ $T_{1.4}$
70
READ REQUESTS
DATA ACCESS PROGRAM
60

FIG. 5


147
MONITOR
FIG. 6
LOCAL AREA NETWORK
151
WIDE AREA NETWORK
152
REMOTE COMPUTER
149b
150b
APPLICATION PROGRAMS
136b
REMOTE COMPUTER
149a
150a
APPLICATION PROGRAMS
136a
140
KEYBOARD
154
MODEM
NETWORK INTERFACE
153
146
SERIAL PORT INTERFACE
142
123
VIDEO ADAPTER 148
134
OPTICAL DRIVE INTERFACE
131
130
SYSTEM BUS
133
MAGNETIC DISK DRIVE INTERFACE
129
128
132
HARD DISK DRIVE INTERFACE
127
139
20
PROCESSING UNIT 121
122
SYSTEM MEMORY
(ROM) 124
BIOS 126
(RAM) 125
OPERATING SYSTEM 135
APPLICATION PROGRAMS136
OTHER PROGRAM MODULES 137
PROGRAM DATA 138
OPERATING SYSTEM 135
APPLICATION PROGRAMS136
OTHER PROGRAM MODULES 137
PROGRAM DATA 138

INCREMENTALLY RESTORING A MASS STORAGE DEVICE TO A PRIOR STATE

RELATED APPLICATION

[0001] This application claims the benefit of U.S. Provisional Patent Application Serial No. 60/257,499, entitled "Methods and Systems for Backing Up and Restoring Computer Data, " filed Dec. 21, 2000, which is incorporated herein by reference.

BACKGROUND OF THE INVENTION

[0002] 1. The Field of the Invention

[0003] The present invention relates to backing up and restoring computer data. More specifically, the present invention relates to systems and methods for minimizing the loss of computer data when restoring computer data that was lost due to the interruption.

[0004] 2. The Prior State of the Art

[0005] With the advent of personal computers, businesses worldwide rely on computer data in performing daily business routines. However, a variety of events such as natural disasters, terrorism, or more mundane events such as computer hardware and/or software failures can occur while a computer is processing. These failures often result in causing the computer data to become corrupt, unreliable, or even lost. The corruption or loss of data, such as customer lists, financial transactions, business documents, business transactions, and so forth, can cause havoc to businesses by resulting in the loss of large investments of time and/or money.

[0006] The loss or corruption of computer data is particularly devastating in the world of electronic commerce. The Internet has allowed individuals all over the world to conduct business electronically, thereby resulting in the continual upload of electronic orders. However, all copies of the orders are electronic and thus corruption or loss of the electronic computer data can result in the loss of the business represented by the lost orders.

[0007] Recognizing the commercial value of reliable computer data, businesses seek ways to protect their data and to reconstruct data that has become corrupt, unreliable, or lost. Traditional approaches of data protection and reconstruction have involved creating a backup copy of the computer data. While it is a simple procedure to preserve a backup copy of an individual file on a floppy disk, the creation of a backup copy becomes more difficult as the amount of data increases.

[0008] Perhaps one of the simplest approaches to creating a backup copy of a large volume of computer data is to copy the data from a mass storage system to an archival device, such as one or more magnetic tapes. This method stores large amounts of computer data at the expense of immediate access to the data. The magnetic tapes are stored either locally or remotely, and the data is copied from the magnetic tapes to the mass storage system when problems arise with the mass storage system.

[0009] While the use of an archival device to preserve data loss has the advantage of being relatively simple and inexpensive, it also has severe limitations. One such limitation is the amount of time that prevents user accessibility to the computer data while a backup copy is created and while data is reconstructed. The prevention of user accessibility has traditionally been required to ensure that no data has changed during the process. Because user inaccessibility of data is undesirable, backup copies are created less frequently, thereby causing the computer data in the backup copy to become stale. Similarly, transferring computer data from a magnetic tape to a mass storage system can become very lengthy because the computer data is transferred on a file-by-file basis. The time is further lengthened when the archival mass storage device is remotely located and is not accessible over a network. These long reconstruction periods result in extended computer inaccessibility and cost businesses increased amounts of time and money.

[0010] Another limitation of the traditional methods is that the backup copy represents data as it existed at a previous instant in time, meaning that the backup copy is not current with subsequent changes made to the original copy. The creation of a backup copy provides the security that a large portion of the computer data can be recovered in the event that the original copy becomes corrupt or lost. This limits the loss to include only the changes made to the original copy since the creation of the last backup copy. However, in some businesses, if the computer data is not current, the data is stale, unreliable, and even useless. This is particularly troubling in the financial world where rates and information change with great frequency. Thus, when the computer data becomes corrupt or lost, the businesses that rely on information that changes frequently are exposed to the risk of losing all of their valuable computer data.

[0011] It would, therefore, represent an advancement in the art to have a system and method for backing up computer data that could further minimize the amount of computer data that is lost due to a computer failure or corruption of data. It would also represent an advancement in the art to have a system that allowed data to be backed up without terminating user access to the mass storage system.

SUMMARY OF THE INVENTION

[0012] The present invention relates to systems and methods for backing up and restoring computer data. The invention enables computer data to be restored in an incremental manner in the event that data is corrupted or lost. In particular, if data is lost or corrupted, the data can be incrementally advanced through successively older states until a valid set of data is obtained. In this manner, data can be restored to a state that is newer than that associated with a full mirrored or archived copy of the data. Thus, full mirror or archiving operations on a volume of data can be less frequent without the risk of losing changes to the volume of data that have occurred since the last full mirror or archiving operation.

[0013] According to the invention, a mass storage device stores a plurality of data blocks at time T_o. At T_o, a mirrored copy of the data stored of the mass storage device may be made and stored such that the data at T_o can be conveniently restored, if necessary. Obtaining a mirrored copy of data is often a time or resource consuming process that is preferably conducted relatively infrequently. In order to preserve the data that is changed after the mirrored copy of data is created and before a potential event causing loss of data, data blocks that are to be overwritten after T_o are stored in a preservation memory. In particular, if, after T_o, a specified data block is

to be overwritten as part of a write operation, a copy of the original data block is stored in a preservation memory prior to the original data block being replaced in the mass storage device. In addition, the data block stored in the preservation memory is time-stamped or otherwise marked so as to designate the time of the write operation or to designate a chronological position of the write operation with respect to other write operations.

[0014] As write operations are successively performed after time T_o, the original data blocks that are to be overwritten are sequentially stored in the preservation memory with an associated time stamp. Thus, the data blocks that are overwritten or otherwise changed after To are preserved in a preservation memory and the time or the order in which the data blocks were overwritten in the mass storage device is specified.

[0015] In the event that certain data blocks in the mass storage unit device are lost or become corrupted, the data blocks stored in the preservation memory can be used to incrementally restore or reconstruct a valid set of data without reverting completely back to the data as it exists at time T_o. If, for example, invalid or corrupted data is written to certain data blocks in the mass storage device after time T_o, the original, valid data blocks are stored in a preservation memory as described above. Using the time stamps specifying the chronological sequence in which the data blocks stored in the preservation memory were overwritten in the mass storage device, the data blocks in the preservation memory are written to the current data stored in the mass storage device.

[0016] After one or more data blocks from the preservation memory are written back to the mass storage device data in reverse chronological order, a valid set of data is eventually obtained at the mass storage device. Thus, the data blocks stored in the preservation memory are used to reconstruct the data without requiring the data to be reverted completely back to the data as it existed at time T_o.

[0017] It is noted that, according to the invention, complete mirror or archive operations performed on the full volume of data stored in the mass storage device can be less frequent than would be otherwise required in the absence of the invention The data blocks stored on preservation memory are used to restored corrupted data to a state more recent than that associated with the most recent full mirrored copy of the data. It is also noted that storing a sequence of overwritten data blocks in the preservation memory is typically less resource intensive than performing a full mirror operation on the data of the mass storage device. In addition, the data stored in the preservation memory in response to the write request can be a data block having fine granularity as opposed to being an entire file. For instance, a data block stored in a preservation memory response to a write operation in the mass storage device can be data associated with a single sector or another portion or segment of the mass storage device. Storing data in a preservation memory in this manner avoids the need of copying entire files in response to write operations and permits the data to be preserved independent of any file structure associated with the data.

[0018] Additional features and advantages of the invention will be set forth in the description that follows, and in part will be obvious from the description, or may be learned by the practice of the invention. The features and advantages of the invention may be realized and obtained by means of the instruments and combinations particularly pointed out in the appended claims. These and other features of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0019] In order that the manner in which the above recited and other advantages and features of the invention are obtained, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments thereof that are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered to be limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

[0020] **FIG. 1** is a block diagram representing the computer system having an associated mass storage device and preservation memory in which the invention can be implemented.

[0021] **FIG. 2** is a block diagram of the computer system of **FIG. 1**, showing data stored in the mass storage device at time T_o.

[0022] **FIG. 3** illustrates a sequence of write operations in which data blocks that are to be overwritten are stored in a preservation memory.

[0023] **FIG. 4** illustrates an operation in which data in a mass storage device is incrementally restored in response to a data corruption event.

[0024] **FIG. 5** illustrates the operation of a virtual device that appears as if it contains a copy of data as it existed previously in a mass storage device.

[0025] **FIG. 6** illustrates a computer system representing an example of a suitable operating environment for the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0026] The present invention relates to backing up and restoring computer data. In particular, the invention enables corrupted data to be restored without requiring the data to be reverted completely back to a most recent full-mirrored copy of the data. Data blocks that are to be overwritten after a point in time in which a mirrored copy of the data has been created are stored in a preservation memory. The data blocks are associated with a time stamp other information that designates the time or the chronological order in which the data blocks were overwritten or the original data blocks in the mass storage device were overwritten. In general, the term "time stamp" refers to any such information designating the chronological order or the time of the data blocks stored in the preservation memory. The data blocks in the preservation memory can be used in the event of data corruption to incrementally roll the current, corrupted copy of the data in the mass storage device to a valid set of data.

[0027] **FIG. 1** illustrates a computer system in which the invention can be practiced. The computer system of **FIG. 1** includes a computer **10**, a mass storage device **12**, a preservation memory **14**, and an I/O device **16**. Computer **10** can be any computer or other processing device that manages, generates, stores, or otherwise processes data. For instance, computer **10** can be a conventional personal computer, a server, a special purpose computer, or the like. **FIG. 6**, discussed in greater detail below, sets forth details of one example of a computer in which the invention can be practiced.

[0028] Mass storage device **12** is associated with computer **10** and is used to store data obtained from computer **10**. In general, mass storage device **12** is a writable, nonvolatile mass storage device. In particular, mass storage device **12** can be the hard drive associated with a conventional personal computer or any other storage volume that is used to store data obtained from computer **10**.

[0029] Preservation memory **14** is a physical or logical device associated with computer **10** in which data blocks that are to be overwritten in mass storage device **12** are stored. Preservation memory **14** can be a volatile device, such as a random access memory (RAM) or any other device that can store data blocks that are to overwritten in mass storage device **12**. Although preservation memory **14** is illustrated as being a separate device in **FIG. 1**, the preservation memory can be a partition or another portion of mass storage device **12**.

[0030] As shown in **FIG. 1**, computer system **10** may include an input/output (I/O) device **16** that allows a user to manipulate data processed by computer **10** and stored in mass storage device **12**. In general, computer **10** can have I/O device **16** or any other peripheral device that enables users to manipulate data. In addition, computer **10** typically includes applications or software having data storage requirements satisfied by the mass storage device **12**. In general, however, the invention is broad enough to extend to substantially any computer **10** and associated hardware and software, so long as there is a mass storage device **12** for storing data and preservation memory **14** for storing overwritten data blocks.

[0031] **FIG. 2** illustrates the computer system of **FIG. 1** at a time T_0, when mass storage device **12** includes a set of data blocks storing given data. In the example of **FIG. 2**, for purposes of illustration, mass storage device **12** is shown as including five data blocks, namely, data blocks A, B, C, D, and E. At T_0, it is assumed that a mirrored or other backup copy of the full volume of data stored in the mass storage device is created and stored. The mirrored or backup copy of the data obtained at T_0 can be generated in any desired manner, including conventional ways. In one embodiment, the mirrored copy or backup of the data at T_0 can be obtained using the techniques disclosed in U.S. Pat. No. 5,649,152, entitled "METHOD AND SYSTEM FOR PROVIDING A STATIC SNAPSHOT OF DATA STORED ON A MASS STORAGE SYSTEM" or U.S. Pat. No. 5,835,953, entitled "BACKUP SYSTEM THAT TAKES A SNAPSHOT OF THE LOCATIONS IN A MASS STORAGE DEVICE THAT HAS BEEN IDENTIFIED FOR UPDATING PRIOR TO UPDATING", which are incorporated herein by reference. In general, U.S. Pat. No. 5,649,152 discloses a technique for taking a snapshot of data as it exists at a selected moment by preserving data overwritten by write operations. U.S. Pat. No. 5,835,953 generally describes techniques for obtaining snapshots or mirrored copies of data by transferring minimal amount of data to a remote location. Alternatively, the mirrored or backup copy of the data of the mass storage device **14** at T_0 can be obtained using conventional data transfer processes.

[0032] Depending on the nature of the data in mass storage device **12**, it may be useful to obtain a mirrored or backup copy of the data at T_0 that includes logically consistent data. This is particularly important when the data stored in mass storage device **12** represents transactions, each requiring a sequence of write operations or otherwise requiring a set of I/O operations to be performed in order to have a valid and useful set of data. Data is logically inconsistent when fewer than all of the necessary operations in a transaction or in another required sequence of S/O operations have been performed. Thus, depending on the nature of the data stored in mass storage device **12**, the mirrored or backup copy of the data at T_0 may need to be logically consistent data.

[0033] Obtaining a mirrored or backup copy of the data at T_0 enables the data blocks A, B, C, D, and E stored at mass storage device **12** at T_0 to be reconstructed simply by accessing the mirrored or backup copy of the data. As stated above, however, full mirror or backup operations are often time or bandwidth intensive, such that relatively infrequent complete mirror or backup operations are desirable. As discussed above, and as described in more detail below, the present invention uses preservation memory **14** to store data blocks that enable data of mass storage device **12** to be incrementally restored in reverse chronological order without requiring the data to be reverted completely back to time T_0. Although periodic mirrored or backup copies of the data of the mass storage device **12** are often useful, the invention can also be practiced solely on the basis of data blocks being stored at preservation memory **14** without periodic mirrored or backup copies.

[0034] In order to describe the manner in which the data is preserved in the preservation memory after time T_0, **FIG. 3** presents a specific example of a set of data blocks in the mass storage device and a sequence of the write operations applied thereto. As shown in **FIG. 3**, the data blocks **20** stored in mass storage device at T_0 are designated as (A, B, C, D, E). At $T_{1.0}$, the computer **10** in **FIG. 2** issues a write request, whereby data block **30** (A_1) is to overwrite existing the data block A, resulting in a set of data blocks **20***a* (A_1, B, C, D, E). In general, in response to a issued write request and prior to the data block in a mass storage device being overwritten with the new data block, the existing data block is stored in the preservation memory, along with a time stamp.

[0035] In this example, at $T_{1.0}$, data block A is written to the preservation memory, along with a time stamp designating the time $T_{1.0}$ as shown at **40***a*. Thus, even though data block A has been overwritten in a mass storage device, the data block A is stored in the preservation memory in the event that this data block is needed to reconstruct corrupted data at some future point.

[0036] At time $T_{1.1}$, data block **30***b* (D_1) is written to the data stored in the mass storage device, resulting in a set of data blocks **20***b* (A_1, B, C, D_1, E). Prior to the write operation, the data block D, which is to be overwritten, is

stored in the preservation memory, along with a time stamp designating the time T_{11}, as shown at **40***b*.

[0037] At time $T_{1.2}$, a data corruption event occurs, resulting in data block **30***c* (D_x) being written to the data stored in the mass storage device. As used in this example, the subscripts "x" and "y" represent corrupted data. The data corruption can have substantially any cause, such as data entry problems, software problems, hardware problems, or the like. The data corruption event at time $T_{1.2}$ results in the set of data blocks **20***c* (A_1, B, C, D_x, E). Prior to the write operation, the data block D_1, which is to be overwritten, is stored in the preservation memory, along with a time stamp designating the time $T_{1.2}$, as shown at **40***c*.

[0038] At time $T_{1.3}$, data block **30***d* (B_1) is written to the data stored in the mass storage device, resulting in a set of data blocks **20***d* (A_1, B_1, C, D_x, E). Prior to the write operation, the data block B, which is to be overwritten, is stored in the preservation memory, along with a time stamp designating the time $T_{1.3}$, as shown at **40***d*.

[0039] Finally, at time $T_{1.4}$, another data corruption event occurs, resulting in data block **30***e* (D_Y) being written to the data stored in the mass storage device. The data corruption event at time $T_{1.4}$ results in the set of data blocks **20***e* (A_1, B_1, C, D_y, E). Prior to the write operation, the data block D_x, which is to be overwritten, is stored in the preservation memory, along with a time stamp designating the time $T_{1.4}$, as shown at **40***e*.

[0040] Thus at time $T_{1.4}$, the data blocks **20***e* existing at the mass storage device are (A_1, B_1, C, D_Y, E). As noted above, the data block D_y represents corrupted data, which potentially makes all of the data stored in mass storage device unusable. At $T_{1.4}$, preservation memory also has stored therein data blocks (A, D, D_1, B, D,) and the corresponding time stamps ($T_{1.0}$, T_{11}, T_{12}, T_{13}, T_{14}).

[0041] Because the data corruption event has occurred since T_o, a valid set of data can be obtain by reverting completely to the data as it existed in the mass storage device at T_o by accessing any mirrored or backup copy of the full volume data that was created at T_o. However, reverting completely to the data as it existed at T_o would result in a loss of all subsequent data written to the mass storage device after T_o. Thus, according to the invention, in response to the data corruption event, the data blocks stored in the preservation memory are used to incrementally restore the data in the mass storage device in reverse chronological order to at a point at which valid, non-corrupted data exists as shown in **FIG. 4**. The preservation memory in **FIG. 4** at time T_2 includes the set of data blocks and corresponding time stamps that existed in the preservation memory at time $T_{1.4}$. Likewise, the mass storage device in **FIG. 4** at time T_2 includes a set of data blocks **20***e* (A_1, B_1, C, D_y, E) that existed at $T_{1.4}$ in **FIG. 3**.

[0042] The data restoration operation illustrated in **FIG. 4** begins with the current set of potentially corrupted data blocks **20***e* and the data blocks **40***e* stored in the preservation memory. The time stamps $T_{1.0}$- $T_{1.4}$ are used to reconstruct the data that previously existed in the mass storage device in reverse chronological order. Thus, the most recently stored data block in preservation memory (D_x) is written to the set of data as it existed at time T_2 to roll back the set of data in the mass storage device to the state in which it existed at $T_{1.3}$, resulting in a set of data (A_1, B_1, C, D_x, E).

[0043] It is noted that this operation of rolling back the data of the mass storage device as illustrated in **FIG. 4** can be performed using the actual data stored in the mass storage device or copy of thereof.

[0044] Upon rolling back to $T_{1.3}$, it is determined, either by the computer or manually, that the set of data **20***d* of T_{13} still represents corrupted data, in that data block D_x is corrupted. Because the data remains corrupted, the data of the mass storage device is further rolled back to time $T_{1.2}$ using data block B of the preservation memory, which has the time stamp T_{13}. Thus, the resulting data representing the state of the mass storage device at $T_{1.2}$ is (A_1, B, C, D_x, E) **20***c*. Because data blocks **20***c* also include corrupted data block D_x, it is determined that the set of data blocks **20***c* represent corrupted data. Accordingly, the data is further rolled back to the state at which it existed at the mass storage device at T_{11} by writing to the data at the mass storage device the data block of the preservation memory that is next in reverse chronological order. In particular, data block D_1 having time stamp $T_{1.2}$ is written to the data blocks of mass storage device, resulting in a set of data blocks **20***b* (A_1, B, C, D_1, E), which represents the data as it existed at the mass storage device at $T_{1.1}$.

[0045] At this point, it is determined that the data **20***b* (A_1, B, C, D_1, E) represents valid, non-corrupted data. Thus, the data blocks of the preservation memory have been used to incrementally restore in reverse chronological order the data blocks of the mass storage device until such time that a valid set of data is obtained. It is also noted that the data blocks **20***b* (A_1, B, C, D_1, E) includes certain data (i.e.$_2$, A_1. and D_1) that would have not been included in the restored data had the data been reverted completely back to the mirrored or backup copy data of T_o . Moreover, this more recent data is restored without requiring a sequence of full mirror or backup operations after T_o .

[0046] In view of the foregoing, the operation for restoring the data generally involves applying the data blocks the preservation memory in reverse chronological order to a current copy of the data blocks of the mass storage device until such time that the data blocks represent valid non-corrupted data.

[0047] As can be understood, when the number of write operations is large or frequent, the number of data blocks stored in the preservation memory can increase rapidly. Thus, in practice, there is a trade-off between the frequency of full mirror or backup operations and the volume of data blocks that are stored in the preservation memory. In one embodiment, the frequency at which the full mirror or backup operations are performed is determined by the frequency at which the preservation memory is filled or reaches a certain size. Other words, as the volume of data stored in the preservation memory approaches the capacity of preservation memory, a full mirror or backup operation is performed on the mass storage device. This enables the data in a preservation memory to be discarded, since it is no longer needed in view of the fact that a more recent mirrored or backup copy of the data of the mass storage device has been created.

[0048] While the invention has been described herein in reference to incrementally restoring corrupted data in reverse chronological order in response to a data corruption event, there are other uses for the basic methods of inven-

tion. For instance, the data blocks of the preservation memory can be used to roll the data in the mass storage device back to previous state for other reasons. Indeed, in substantially any situation in which a user wishes to obtain data as it existed previously in a mass storage device. In yet another embodiment, data stored in the preservation memory is combined with a mirrored or backup copy of data as it existed at a selected point in time to roll the data of the mass storage device back to a time prior to the creation of the mirrored or backup copy. As described above in reference to **FIG. 4**, data blocks stored in a reservation memory prior to being overwritten in the mass storage device can be used to incrementally roll back the data in the mass storage device. Thus, if the preservation memory includes a sequence of data blocks that were overwritten in the mass storage device prior to a subsequent mirrored or backup copy of the data of the mass storage device, the sequence of data blocks in the preservation memory can be used to incrementally advance the mirrored or backup copy back in time as desired.

[0049] With overwritten data blocks stored in the preservation memory, the invention can be used to establish a virtual mass storage device ("virtual device") that permits data that existed previously in the mass storage device to be accessed. The virtual device **50** of **FIG. 5** appears, from the standpoint of the user or an application (i.e., data access program **60**) that accesses the virtual device, to contain the data that existed at a previous point in time. For example, the virtual device **50** can be accessed using an operating system and an associated file system as if the virtual device were another hard drive installed on computer **10** of **FIG. 1**.

[0050] One example of the manner in which the virtual device **50** can be used to access data as it existed at a previous point in time is illustrated in **FIG. 5**, using the same set of data that has been previously described above in reference to **FIGS. 3 and 4**. In the example of **FIG. 5**, it has been determined that a valid set of data can be obtained by rolling the data blocks **20***e* stored in the mass storage device **12** to the state in which they existed at $T_{1.1}$ using the data blocks **40***e* stored in preservation memory **14**, as has been described in reference to **FIG. 4**.

[0051] When virtual device **50** is used to access data, data access program **60** issues read requests **70** to virtual device **50** rather than addressing the requests specifically to mass storage device **12** or preservation memory **14**. In this example, the read requests are used to access the most recent valid set of data that existed prior to the data corruption event that introduced corrupted data blocks D_x and D_y to the set of data blocks in mass storage device **12**. As described previously, it can be determined that a set of valid data that existed just prior to the introduction of corrupted data block D_x can be obtained. In other words, the valid set of data at $T_{1.1}$, which is just prior to the data corruption event of $T_{1.2}$, can be obtained.

[0052] Upon receiving a request for data, as it existed in the previous, non-corrupted state, the virtual device determines whether the data satisfying the request is to be obtained from mass storage device **12** or preservation memory **14**. If the read request is directed to a data block having a non-corrupted version that has been stored in preservation memory **14** at or after the data corruption event (i.e., at or after $T_{1.2}$), the oldest such data block is accessed to respond to the read request. For example, if data access program **60** requests a data block at the "D" position that is associated with the most recent set of valid data, the read request is processed by accessing the oldest non-corrupted data block stored in the preservation memory at or after $T_{1.2}$. Thus, in response to the request for the data block "D", the virtual device accesses data block D_1, shown in crosshatch in **FIG. 5**. In a similar manner, a request for the "B" data block is processed by accessing data block B stored in preservation memory **14**, which is also shown in crosshatch in **FIG. 5**.

[0053] If, however, the read request is directed to a data block that does not have a non-corrupted version stored in preservation memory **14** at or after the data corruption event (i.e., at or after $T_{1.2}$), the corresponding data block is accessed from mass storage device **12** to respond to the read request. For instance, requests for data blocks at positions "A", "C", and "E" are processed by accessing the corresponding data blocks A_1, C, and E from mass storage device **12**, which are shown in crosshatch in **FIG. 5**. In this way, data access program **60** can request the full set of data blocks that existed at the previous, non-corrupted state from virtual device **50** and receive, in response thereto, the set of data blocks (A_1, B, C , D_1, and E). Virtual device **50** identifies which data blocks to obtain and whether to obtain the data blocks from mass storage device **12** or preservation memory **14**. Using the data blocks, data access program **60** does not need to know the details of mass storage device **12**, preservation memory **14**, or the data blocks stored thereon, but instead simply issues a request to virtual device **50** as if virtual device **50** contained the prior set of data.

[0054] In one embodiment, the prior set of data can be reconstructed in the manner set forth above in reference to **FIG. 5**. Specifically, the prior set of non-corrupted data can be obtained by establishing a virtual device and reading the prior set of non-corrupted data as set forth above. Data access program **60** then makes the prior set of non-corrupted data available as needed for substantially any use.

[0055] The embodiments of the present invention may comprise a special-purpose or general-purpose computer that includes various components, as discussed in greater detail below. Embodiments within the scope of the present invention may also include computer-readable media for carrying or having computer-executable instructions or data structures stored thereon. Such computer-readable media can be any available media that can be accessed by a general-purpose or special-purpose computer. By way of example, and not limitation, such computer-readable media can comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to carry or store desired program code means in the form of computer-executable instructions or data structures and which can be accessed by a general-purpose or special-purpose computer.

[0056] When information is transferred or provided over a network or another communications connection (either hardwired, wireless, or a combination of hardwired or wireless) to a computer, the computer properly views the connection as a computer-readable medium. Thus, any such connection is properly termed a computer-readable medium. Combinations of the above should also be included within

the scope of computer-readable media. Computer-executable instructions comprise, for example, instructions and data which may cause a general-purpose computer, special-purpose computer, or special-purpose processing device to perform a certain function or group of functions.

[0057] **FIG. 6** and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. Although not required, the invention can be described in the general context of computer-executable instructions, such as program modules, being executed by computers in network environments. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Computer-executable instructions, associated data structures, and program modules represent examples of the program code means for executing steps of the methods disclosed herein. The particular sequence of such executable instructions or associated data structures represents examples of corresponding acts for implementing the functions described in such steps.

[0058] Those skilled in the art will appreciate that the invention may be practiced in network computing environments with many types of computer system configurations, including personal computers, hand-held devices, mobile telephones, personal digital assistants ("PDAs"), multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like, one example having been presented in **FIG. 1**. The invention may also be practiced in distributed computing environments where local and remote processing devices are linked (either by hardwired links, wireless links, or by a combination of hardwired or wireless links) through a communications network and both the local and remote processing devices perform tasks.

[0059] With reference to **FIG. 6**, an example system for implementing the invention includes a general-purpose computing device in the form of a computer **120**, including a processing unit **121**, a system memory **122**, and a system bus **123** that couples various system components including system memory **122** to processing unit **121**. Computer **120** and the associated component illustrated in **FIG. 6** represent a more detailed illustration of computer **10** of **FIG. 1**. System bus **123** may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. System memory may include read only memory ("ROM") **124** and random access memory ("RAM") **125**. A basic input/output system ("BIOS") **126**, containing the basic routines that help transfer information between elements within the computer **120**, such as during start-up, may be stored in ROM **124**.

[0060] Computer **120** may also include a magnetic hard disk drive **127** for reading from and writing to a magnetic hard disk **139**, a magnetic disk drive **128** for reading from or writing to a removable magnetic disk **129**, and an optical disk drive **130** for reading from or writing to removable optical disk **131** such as a CD-ROM or other optical media. Magnetic hard disk drive **127**, magnetic disk drive **128**, and optical disk drive **130** are connected to system bus **123** by a hard disk drive interface **132**, a magnetic disk drive-interface **133**, and an optical drive interface **134**, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer-executable instructions, data structures, program modules and other data for computer **120**. Although the example environment described herein employs a magnetic hard disk **139**, a removable magnetic disk **129** and a removable optical disk **131**, other types of computer readable media for storing data can be used, including magnetic cassettes, flash memory cards, digital versatile disks, Bernoulli cartridges, RAMs, ROMs, and the like.

[0061] Program code means comprising one or more program modules may be stored on hard disk **139**, magnetic disk **129**, optical disk **131**, ROM **124**, or RAM **125**, including an operating system **135**, one or more application programs **136**, other program modules **137**, and program data **138**. A user may enter commands and information into computer **120** through keyboard **140**, pointing device **142**, or other input devices (not shown), such as a microphone, joy stick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to processing unit **121** through a serial port interface **146** coupled to system bus **123**. Alternatively, the input devices may be connected by other interfaces, such as a parallel port, a game port or a universal serial bus ("USB"). A monitor **147** or another display device is also connected to system bus **123** via an interface, such as video adapter **148**. In addition to the monitor, personal computers typically include other peripheral output devices (not shown), such as speakers and printers.

[0062] Computer **120** may operate in a networked environment using logical connections to one or more remote computers, such as remote computers **149*a*** and **149*b***. Remote computers **149*a*** and **149*b*** may each be another personal computer, a server, a router, a network PC, a peer device or other common network node. Remote computers **149*a*** and **149*b*** may include many or all of the elements described above relative to the computer **120**, although only memory storage devices **150*a*** and **150*b*** and their associated application programs **136*a*** and **136*b*** are illustrated in **FIG. 6**. The logical connections depicted in **FIG. 6** include a local area network (LAN) **151** and a wide area network (WAN) **152** that are presented here by way of example and not limitation. Such networking environments are commonplace in office-wide or enterprise-wide computer networks, intranets and the Internet.

[0063] When used in a LAN networking environment, computer **120** is connected to local network **151** through a network interface adapter **153** or similar adapter. When used in a WAN networking environment, computer **120** may include a modem **154**, a wireless link, or other means for establishing communications over the wide area network **152**, such as the Internet. Modem **154**, which may be internal or external, is connected to system bus **123** via the serial port interface **146**. In a networked environment, program modules depicted relative to computer **120**, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing communications over wide area network **152** may be used.

[0064] The present invention may also be embodied in other specific forms without departing from its spirit or

essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes, which come within the meaning and range of equivalency of the claims, are to be embraced within their scope.

What is claimed and desired to be secured by United States Letters Patent is:

1. In a computer system that includes a mass storage device that stores a plurality of data blocks, a method of updating the data blocks while enabling a previous state of the mass storage device to be accessible, comprising the acts of:

receiving a write request that is to overwrite an existing data block in the mass storage device with a new data block;

prior to executing the write request, storing a copy of the existing data block in a preservation memory associated with the computer system and associating a time stamp with the copy of the data block in the preservation memory; and

executing the write request, such that the existing data block stored in the mass storage device is overwritten with the new data block.

2. A method as recited in claim 1, wherein the preservation memory comprises a volatile memory device.

3. A method as recited in claim 1, wherein the preservation memory comprises a portion of the mass storage device.

4. A method as recited in claim 1, further comprising the act of creating a backup copy of the plurality of data blocks stored by the mass storage device prior to the act of receiving the write request.

5. A method as recited in claim 6, wherein the act of creating the backup copy is conducted when the plurality of data blocks represent logically consistent data.

6. A method as recited in claim 1, further comprising the act of restoring the plurality of data blocks stored at the mass storage device to a state in which the plurality of data blocks existed at a previous point in time using the copy of the existing data block stored by the preservation memory.

7. A method as recited in claim 1, further comprising the act of storing a sequence of existing data blocks in the preservation memory in response to one or more write requests that are to overwrite the existing data blocks at the mass storage device and associating a time stamp with each of the existing data blocks of the sequence.

8. A method as recited in claim 7, further comprising the act of restoring the plurality of data blocks stored at the mass storage device to a state in which the plurality of data blocks existed at a previous point in time using the sequence of existing data blocks stored in the preservation memory.

9. A method as recited in claim 8, further comprising the act of experiencing a data corruption event in the mass storage device, wherein the act of restoring the plurality of data blocks is conducted to restore the plurality of data blocks to obtain non-corrupted data.

10. A method as recited in claim 8, wherein the act of restoring comprises the act of applying the data blocks of the sequence of existing data blocks stored in the preservation memory to a current version of the plurality of data blocks stored at the mass storage device in reverse chronological order.

11. A method as recited in claim 9, wherein the act of restoring the plurality of data blocks is conducted to restore the plurality of data blocks to obtain non-corrupted data and wherein the act of applying the data blocks comprises the act of determining when the non-corrupted data is obtained.

12. A method as recited in claim 1, further comprising the act of establishing a virtual device at the computer system that appears as if it contained the plurality of data blocks stored at the mass storage device in a state in which the plurality of data blocks existed at a previous point in time.

13. A method as recited in claim 12, further comprising the acts of:

receiving, at the virtual machine, a read request specifying a requested data block; and

in response to the read request:

if the preservation memory includes a copy of a non-corrupted version of the requested data block that was stored in the preservation memory at or after the previous point in time, processing the read request using the oldest such non-corrupted version of the requested data block stored in the preservation memory; and

if the preservation memory does not include said copy of a non-corrupted version of the requested data block that was stored in the preservation memory at or after the previous point in time, processing the read request using a version of the requested data block stored in the mass storage device.

14. In a computer system that includes a mass storage device that stores a plurality of data blocks, a method of restoring the data blocks of the mass storage device to a previous state, comprising the acts of:

iteratively performing the acts of:

receiving a write request that is to overwrite an existing data block in the mass storage device with a new data block;

prior to executing the write request, storing a copy of the existing data block in a preservation memory associated with the computer system and associating a time stamp with the copy of the data block in the preservation memory; and

executing the write request, such that the existing data block stored in the mass storage device is overwritten with the new data block;

experiencing a data corruption event whereby at least one of the plurality of data blocks stored in the mass storage device becomes corrupted; and

restoring the data blocks of the mass storage device to a previous state by applying the copies of the existing data blocks from the preservation memory to the plurality of data blocks of the mass storage device until the previous state is obtained.

15. A method as recited in claim 14, wherein the reverse chronological order is identified using the time stamps associated with the copies of the existing data blocks.

16. A method as recited in claim 14, further comprising the act of creating a backup copy of the plurality of data blocks stored by the mass storage device prior to iteratively performing the act of receiving a write request.

17. A method as recited in claim 16, wherein the act of restoring the data blocks is conducted such that the previous state represents non-corrupted data and is more recent than the backup copy.

18. A method as recited in claim 14, wherein the iteratively performed act of storing a copy of the existing data block is performed on the data block level and independent of any file structure associated with the mass storage device.

19. A method as recited in claim 18, wherein the plurality of data blocks represent data stored in specified sectors of the mass storage device.

20. In a computer system that includes a mass storage device that stores a plurality of data blocks, a method of establishing a virtual device that enables access to the plurality of data blocks as the data blocks existed at the mass storage device at a previous point in time, comprising the acts of:

storing a current version of the plurality of data blocks in the mass storage device;

maintaining, at a preservation memory associated with the computer system, copies of previous versions of data blocks that have since been overwritten at the mass storage device in response to write requests, the previous versions of the data blocks being associated with time stamps specifying a chronological order in which the data blocks were overwritten;

providing access to the current version and the copies of the previous version through a virtual device, wherein the virtual device, in response to read request specifying a particular data block as it existed at the previous point in time, determines whether to access the current version of the data block from the mass storage device or the previous version of the data block from the preservation memory.

21. A method as recited in claim 20, wherein:

the current version of the plurality of data blocks includes one or more corrupted data blocks; and

at the previous point of time, the plurality of data blocks were non-corrupted.

22. A method as recited in claim 20, wherein, in response to the read request, the virtual device performs the acts of:

if the preservation memory includes a copy of a non-corrupted version of the requested data block that was stored in the preservation memory at or after the previous point in time, processing the read request using the oldest such non-corrupted version of the requested data block stored in the preservation memory; and

if the preservation memory does not include said copy of a non-corrupted version of the requested data block that was stored in the preservation memory at or after the previous point in time, processing the read request using a version of the requested data block stored in the mass storage device.

23. A method as recited in claim 20, further comprising experiencing a data corruption event whereby one or more current version of a data block represents corrupted data, wherein the virtual device providing access to the plurality of data blocks as the data blocks existed at the mass storage device at a previous point in time in a non-corrupted state.

24. A method as recited in claim 20, wherein the preservation memory comprises a portion of the mass storage device.

25. A computer system that provides access to data blocks as the data blocks existed at a previous point in time, comprising:

a mass storage device that stores a plurality of data blocks that can be overwritten in response to a write request;

a preservation memory that receives and stores a copies of the data blocks that are to be overwritten prior to the write request being processed and that associates a time stamp with each said copy of a data block; and

a virtual device that provides access to a current version of the data blocks stored at the mass storage device and the copies of the data blocks at the preservation memory, wherein the virtual device, in response to read request specifying a particular data block as it existed at the previous point in time, determines whether to access a current version of the specified data block from the mass storage device or a copy of the specified data block from the preservation memory.

26. A computer system as recited in claim 25, wherein the preservation memory comprises a portion of the mass storage device.

27. A computer system as recited in claim 25, wherein the virtual device, from the standpoint of a data access program, appears as if it contained the data blocks as the data blocks existed in the mass storage device at the previous point in time.

28. A computer system as recited in claim 25, wherein the virtual device, in response to the read request, performs the acts of:

if the preservation memory includes a copy of a non-corrupted version of the specified data block that was stored in the preservation memory at or after the previous point in time, processing the read request using the oldest such non-corrupted version of the specified data block stored in the preservation memory; and

if the preservation memory does not include said copy of a non-corrupted version of the specified data block that was stored in the preservation memory at or after the previous point in time, processing the read request using a version of the specified data block stored in the mass storage device.

29. A computer system as recited in claim 26, further comprising means for creating a backup copy of the data prior to receiving the write request.

* * * * *